UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería en Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

RECONOCIMIENTO AUTOMÁTICO DE IDIOMA MEDIANTE REDES NEURONALES

Ricardo Gómez García
Tutor: Alicia Lozano Diez
Ponente: Joaquín González Rodríguez

Junio 2018

Reconocimiento Automático de Idioma mediante Redes Neuronales

AUTOR: Ricardo Gómez García
TUTOR: Alicia Lozano Diez

Audio, Data Intelligence and Speech (Audias - ATVS)
Dpto. Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2018

Resumen

En este Trabajo Fin de Grado se presenta una comparación entre el rendimiento de cinco sistemas diferentes basados en redes neuronales compuestas principalmente por capas *Long Short Term Memory* (LSTM), para la tarea de la clasificación automática de idioma a través de secuencias de voz de 3 segundos.

Para ello se ha utilizado la base de datos proporcionada por el *National Institute of Standards and Technology* (NIST) *Language Recognition Evaluation* (LRE) en 2009, de la cual se ha extraído información correspondiente a 8 idiomas objetivos que cuentan con una cantidad de datos igual o superior a 200 horas.

Las características de entrada empleadas en los sistemas han sido MFCC, MFCC junto con sus derivadas tanto de primer como de segundo orden y *bottlenecks* extraídos de una red entrenada para clasificación de unidades fonéticas.

La metodología seguida en el proceso del diseño de los diferentes experimentos ha sido empezar con un modelo simple y posteriormente ir aumentando la complejidad según los resultados obtenidos.

Para la implementación de cada una de las redes se ha hecho uso principalmente de las librerías Keras y TensorFlow, las cuales son ampliamente utilizadas actualmente en el desarrollo de sistemas de *machine learning*. Keras actúa por encima de TensorFlow y permite una rápida experimentación a alto nivel y la posibilidad de ejecutar programas sobre GPUs.

La evaluación del rendimiento se ha realizado a través de las medidas de *accuracy* y C_{avg}, las cuales permiten obtener una medida de referencia adecuada en problemas de clasificación de clases balanceadas como es el caso presente en este trabajo.

El punto más relevante que se ha podido extraer a través de la realización de los diferentes experimentos ha sido que al utilizar los *bottlenecks* mencionados anteriormente como características de entrada, se ha conseguido una mejora relativa muy considerable (~36%) en comparación con los MFCC para la tarea de la identificación de idioma. La arquitectura que ha alcanzado un mejor rendimiento ha sido la compuesta por dos capas ocultas LSTM y una capa *feed-forward* a continuación, cada una de ellas compuesta por 512 unidades.

Palabras clave

LSTM, MFCC, *bottlenecks*, Keras, TensorFlow, *machine learning*, redes neuronales, identificación automática de idioma.

Abstract

This Final Degree Thesis presents a comparison between the performance of five different systems based on neural networks composed mainly of Long Short Term Memory (LSTM) layers, for the task of automatic language identification through 3-second voice sequences.

For this purpose, information –regarding 8 different objective languages that amounts to a data quantity equal or higher than 200 hours– has been extracted from the National Institute of Standards and Technology (NIST) Language Recognition Evaluation (LRE) 2009 database.

The input features used in the systems have been MFCC, its first and second level derivatives, and bottlenecks extracted from a trained network for classification of phonetic units.

The methodology used to design the different experiments has consisted in starting out with a simple model and, depending on the results, increasing its complexity.

Each network has been implemented using Keras and TensorFlow libraries, which are broadly used in developing machine learning systems. Keras runs on top of TensorFlow and allows fast high-level experimentation and the possibility to run models on GPUs.

The performance of each network has been tested according to the accuracy and Cavg measurements, which allow the establishment of a reference measure suitable for classification problems with balanced classes, as is the case in this project.

The most relevant aspect that has been extracted from the experiments conducted is that using the bottlenecks as input features show a remarkable relative improvement (~36%) at language identification in comparison with MFCC features. The design that performed the best comprises two LSTM hidden layers followed by one feed-forward layer, each of them consisting of 512 units.

Keywords

LSTM, MFCC, bottlenecks, Keras, TensorFlow, machine learning, neural networks, automatic Language identification.

Agradecimientos

Echando la mirada atrás en este instante se me vienen de golpe tantos momentos e historias únicas vividas a lo largo de esta carrera…sin duda ha sido un extenso camino en donde he encontrado mucho más que unos compañeros de clase, he encontrado a amigos (de los de verdad) para toda la vida.

En primer lugar me gustaría agradecer a mi tutora Alicia, la cual me ha dado la gran oportunidad de hacer este TFG en el único campo de la carrera que realmente me ha gustado y me ha motivado. Aprecio muchísimo además la excelente atención y ayuda proporcionada en todo momento, es un ejemplo a seguir como tutora.

En segundo lugar a mi familia, en especial a mi madre y a mi abuela, las cuales se han desvivido en todo momento por mí desde que tengo uso de razón y siempre han estado cuidándome, estoy seguro de que ahora estarán mucho más orgullosas (mucho más que yo).

A lo más importante que me llevo de la carrera, a aquellos compañeros con los que compartí tantos y tantos momentos, sobre todo risas, sin ellos este gran esfuerzo perdería todo el sentido. No me cabrían todos los nombres en esta página, pero sin duda quiero agradecer a Víctor, que con su positividad y sabiduría se convirtió en una de esas personas que siempre quiero tener a mi lado en todo momento. A Garay, con el he tenido el placer de reírme tantísimo y de una forma que pocas personas logran comprender. A Sergio que fue la primera persona que conocí por aquel primer año cuando éramos unos freshmen y aunque últimamente nos veamos menos, siempre he disfrutado de compartir cualquier tipo de momentos con él, aquel viaje a Mallorca… A Raúl, que juntos hemos arrasado innumerables noches de fiesta y con el que siempre tienes unas buenas risas aseguradas. También a Alfredo, Gonzalo, Claudia, Ana, Pati y una innumerable lista de personas con las que he pasado enormes cantidades de tiempo.

A mis amigos fuera de la universidad, los cuales sin duda alguna me han servido de gran apoyo moral y alguno hasta casi ha colado como Teleco de andar tanto con el grupo de la uni y con el cual siempre es un placer compartir unas buenas sardinas con limón. En especial quiero agradecer a Jujo, que pocas personas como él hay en este mundo sin duda alguna y le agradezco eternamente el estar siempre en todo momento, ya sabe que para mi es como un hermano de otra madre. También por supuesto a Cuevas, Angelito, Richi, Kike y Razvan, grandes personas y amigos desde el instituto.

De corazón, muchísimas gracias a todos.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

1.1 Motivación

La identificación automática de idioma (*Language Identification*, LID) [1] es el proceso de utilizar una computadora para identificar mediante un sistema de forma automática el idioma que está siendo hablado en un segmento de audio.

Las aplicaciones LID [2] se pueden dividir en dos categorías, por un lado, se encuentran aquellas que son utilizadas como preprocesamiento para otras máquinas y por otro lado se encuentran las que son utilizadas como preprocesamiento para oyentes humanos.

Si se analizan aplicaciones actuales como Siri, Google Assistant, o Cortana, que están basadas en el reconocimiento de la voz, se puede ver como estos sistemas para su correcto funcionamiento requieren previamente una entrada manual por parte del usuario que especifique el idioma que va a ser hablado. Un sistema previo de preprocesamiento correspondiente a la primera categoría antes mencionada evitaría por completo esta tarea adicional de la previa especificación del idioma por parte del usuario, haciendo aún más rápido y sencillo el uso de dichas aplicaciones [3].

Por otra parte, también se pueden encontrar sistemas de redireccionamiento de llamadas para que estas puedan ser atendidas por diferentes oyentes. Si analizamos como ejemplo un sistema de redireccionamiento de llamadas de emergencia, donde el tiempo empleado es un punto crítico, se puede ver que la mayoría de estos sistemas se basan en prueba y error, y en muchas ocasiones se realizan múltiples redireccionamientos hasta que el intérprete final es alcanzado [2]. Una aplicación perteneciente a la segunda categoría resultaría en una disminución considerable del tiempo empleado en muchos escenarios y por lo tanto en una ejecución mucho más eficiente y eficaz de la tarea realizada.

Los casos anteriores solo representan dos ejemplos de aplicaciones LID. En la actualidad, dichas aplicaciones son empleadas diariamente en numerosos casos de uso.

Recientemente, la utilización de diferentes tipos de redes neuronales en aplicaciones LID ha logrado superar a la aplicación de técnicas tradicionales del estado del arte, las cuales están basadas principalmente en el modelado de características acústicas. Motivado por el sistema propuesto en [4], en donde se propone el empleo de redes LSTMs para esta tarea y en donde se logra superar el rendimiento de un sistema tradicional en un 26% aproximadamente, en este trabajo se propone la implementación y comparación de diferentes experimentos, los cuales emplean principalmente redes compuestas por capas LSTMs para la identificación de idioma en secuencias de 3 segundos y en donde se utilizan dos tipos de características de entrada a los sistemas descritas más adelante en este trabajo.

1.2 Objetivos

Este Trabajo de Fin de Grado tiene como objetivo principal implementar y comparar el rendimiento de sistemas basados en redes neuronales compuestas de diferentes arquitecturas para la tarea de identificar 8 idiomas a través de secuencias de voz de 3 segundos de duración.

Más detalladamente los objetivos son:

- Investigar la utilización de los sistemas tradicionales acústicos de identificación de idiomas.
- Introducir el funcionamiento de algunos de los diferentes modelos basados en redes neuronales.
- Analizar algunas de las herramientas más utilizas en este momento para abordar problemas de *machine learning.*
- Diseñar e implementar cinco experimentos propuestos basándose en los resultados obtenidos en cada uno de ellos.
- Presentar una comparación justificada entre el rendimiento de dichos sistemas, basándose para ello principalmente en las medidas del *accuracy* y el C_{avg}.

1.3 Organización de la memoria

Con el fin de lograr una mejor estructuración del proyecto, la memoria se ha divido en las siguientes secciones:

- Introducción: Se expone de forma resumida la motivación, el sistema propuesto y los objetivos del trabajo.

- Estado del arte: Se exploran los principales sistemas tradicionales acústicos de reconocimiento automático de idioma y se introduce el funcionamiento de las redes neuronales, así como la motivación para el empleo de diferentes arquitecturas.

- Tecnologías a utilizar: Se analizan las herramientas que se han elegido y que se encuentran entre las más utilizas actualmente para la implementación de sistemas de *machine learning* y que van a ser utilizadas en este trabajo.

- Diseño: Se establece la metodología seguida a la hora de diseñar cada experimento. Además, se especifican las características de entrada, parámetros y arquitectura de las redes implementadas.

- Desarrollo: Se describe el proceso llevado a cabo durante la implementación de los diferentes sistemas previamente diseñados.

- Pruebas y resultados: Se presenta una comparación razonada del rendimiento de los diferentes sistemas implementados.

- Conclusiones y trabajo futuro: Se exponen los puntos más relevantes extraídos de la realización de este trabajo, así como de algunos de los próximos pasos que se pueden llevar a cabo.

2 Estado del arte

2.1 Sistemas tradicionales acústicos de identificación automática de idioma

2.1.1 Fundamentos del proceso de identificación de idioma

Antes de empezar con la descripción de los sistemas tradicionales en esta sección, es conveniente analizar algunos puntos claves presentes en el proceso de identificación de idioma que resultan muy relevantes a la hora de diseñar e implementar cualquier sistema LID.

Existen cuatro características principales [2] que permiten diferenciar un idioma de otros, esto es, una "firma" única y exclusiva de cada idioma. A continuación, se listan y describen de forma resumida estas características:

- Fonemas: Los fonemas difieren entre idiomas, ya que incluso teniendo en cuenta que varios fonemas pueden ocurrir en distintos idiomas, la frecuencia de aparición de estos fonemas es distinta para cada idioma.

- Prosodia: Para cada idioma, las características como la duración, la entonación y los acentos también varían.

- Morfología: Se refiere a las reglas que gobiernan las combinaciones de fonemas para formar las palabras. Existe una gran variedad de reglas que difieren para cada idioma.

- Sintaxis: La forma de combinar las palabras o elementos de una frase también varía entre idiomas. Incluso si dos idiomas contienen la misma palabra, estas se pueden combinar de diferente forma con otros elementos de la frase.

Los sistemas LID explotan una o varias de estas características.

El objetivo presente es identificar el idioma hablado en secuencias de corta duración, y para ello se va a hacer uso de las características acústicas presentes en el segmento de audio. Es por ello por lo que más adelante en esta sección, se hará un breve repaso de los principales sistemas tradicionales basados en características acústicas.

2.1.2 Coeficientes Cepstrales en la escala de Mel (MFCC)

La extracción de características a utilizar (también conocida como parametrización) es una tarea fundamental en el diseño e implementación de cualquier sistema LID, la cual afecta de forma muy significativa el rendimiento del sistema [5]. La parametrización permite representar la información relevante de un segmento de voz de forma compacta mediante una secuencia de vectores de características que contienen información de dicha señal
.
Uno de los métodos más utilizados para extraer características acústicas se realiza a través del cómputo de los Coeficientes Cepstrales en la escala de Mel (*Mel Frequency Cepstral Coefficients*, MFCC). Esta técnica de extracción de características es una de las más empleadas dentro de las técnicas basadas en el dominio de la frecuencia utilizando la escala de Mel, la cual se aproxima a la escala empleada por el oído humano para la percepción del sonido.

La razón principal por la que los MFCC son tan populares es la robustez que presentan al ruido introducido tanto por los hablantes como por las condiciones de grabación.

A continuación [6] [7], se verá el proceso de extracción de los MFCC y para ello se seguirá el diagrama de bloques mostrado en la Figura 2-1:

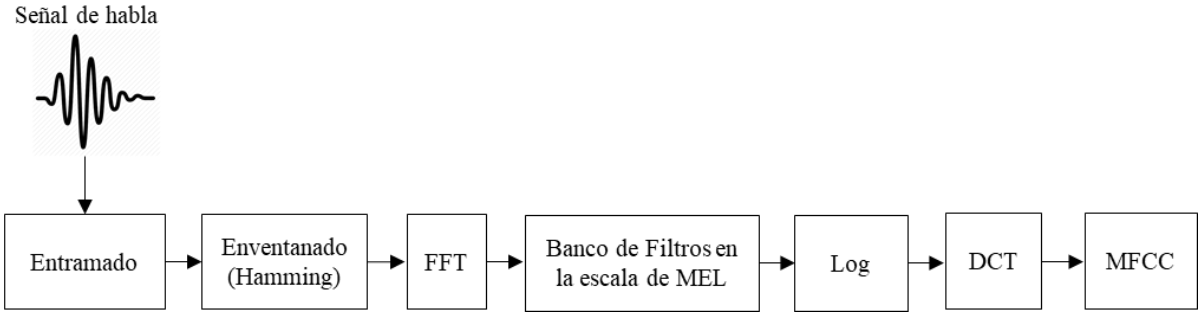


Figura 2-1: Diagrama de bloques que muestra el proceso de extracción de los MFCC.

El primer paso es dividir la señal de audio en diferentes tramas, generalmente constan de una duración aproximada de 20 ms, con un solapamiento de 10 ms para que la transición entre estas sea más suave. Seguidamente se realiza el proceso de enventanado, el cual consiste en multiplicar cada trama por una ventana *Hamming* para suavizar o eliminar las discontinuidades en los extremos de la trama. Posteriormente se calcula para cada trama la Transformada Rápida de Fourier (*Fast Fourier Transform*, FFT), con el objetivo de extraer las componentes frecuenciales de la señal de una forma rápida. El siguiente paso es aplicar el banco de filtros de la escala de Mel, que consiste en una serie de filtros paso-banda triangulares similares a los que se muestran en la Figura 2-2. La escala de Mel es aproximadamente linear hasta 1KHz y logarítmica en frecuencias superiores, e intenta imitar las bandas críticas del sistema auditivo humano. La salida de estos filtros es denominada como los coeficientes de Mel. El último paso es calcular la Transformada Discreta del Coseno (*Discrete Cosine Transform*, DCT) de los coeficientes anteriores, la cual concentra la energía y pasa cada trama al dominio cepstral.

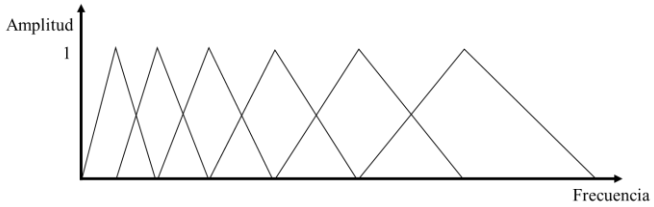


Figura 2-2: Banco de filtros de la escala de Mel.

Los coeficientes resultantes de realizar todo el proceso anterior se denominan características MFCC.

Hay que tener en cuenta que durante el proceso descrito anteriormente puede haber variaciones, como por ejemplo el número de filtros de la escala de Mel empleados.

Una vez visto el proceso de extracción de las características acústicas, se pasará a ver algunos de los principales sistemas tradicionales acústicos de identificación de idioma, cuya evolución ha seguido generalmente la de los sistemas tradicionales acústicos de identificación de locutor, que se adaptan en numerosas ocasiones para la tarea de LID.

2.1.3 Modelos de Mezclas de Gaussianas (GMMs)

Años atrás, los Modelos de Mezclas de Gaussianas (*Gaussian Mixture Models*, GMMs) [8] han sido ampliamente usados para el modelado en aplicaciones de procesamiento de la señal de voz independientes del texto.

El enfoque que sigue un GMM es modelar la distribución de los vectores de características multi-dimensionales extraídas de una secuencia de habla, como pueden ser MFCC. Un GMM está caracterizado por una función de densidad de probabilidad, resultante de realizar una combinación lineal ponderada de múltiples componentes Gaussianas a partir de cada muestra [9]. Cada vector de características cuenta como una muestra y a partir del conjunto de muestras se crea un GMM.

En el caso de reconocimiento de idioma, a partir de un GMM entrenado para cada idioma, podemos calcular la probabilidad de que un vector de características proveniente de una secuencia de habla haya sido generado por un determinado modelo y por lo tanto pertenezca a un determinado idioma.

La desventaja principal de los GMMs es que solo tienen en cuenta la información presente en los datos de entrenamiento, por lo que se requiere un gran volumen de datos proveniente de una gran variedad de fuentes para conseguir un resultado aceptable, de forma contraria se producirá un sobreajuste del modelo, especialmente cuando el número de componentes Gaussianas es alto, como ocurre con los utilizados para reconocimiento de idioma.

2.1.4 Modelo de Referencia Universal (UBM)

El Modelo de Referencia Universal, en inglés *Universal Background Model* (UBM) y también conocido como GMM-UBM, surgió para solventar la problemática de escasez de datos en los modelos GMM.

Un UBM es un GMM que se entrena a partir de una gran cantidad de datos correspondientes a diversos idiomas, de esta forma se crea un modelo universal que representa la distribución de características de secuencias de habla independientemente del idioma. A la hora de entrenar un UBM, hay que prestar especial atención a que los datos a partir de los cuales ha sido entrenado estén balanceados, de forma contraria no conseguiremos modelar un espacio que represente de forma general a todos los subgrupos dentro de los datos. Ejemplos de estos subgrupos pueden ser número de muestras para cada idioma, condiciones en las que fueron grabados y género de los hablantes [8].

A partir de un UBM se crea entonces un GMM adaptado para cada idioma. El proceso de adaptación se realiza típicamente a través del algoritmo Máximo a Posteriori (MAP), donde generalmente se adaptan solo las medias. De esta forma, cada GMM adaptado contiene también información del UBM, dándonos la información particular de cada idioma con respecto al modelo universal, lo que resulta en un sistema con mayor robustez a la falta de datos [10].

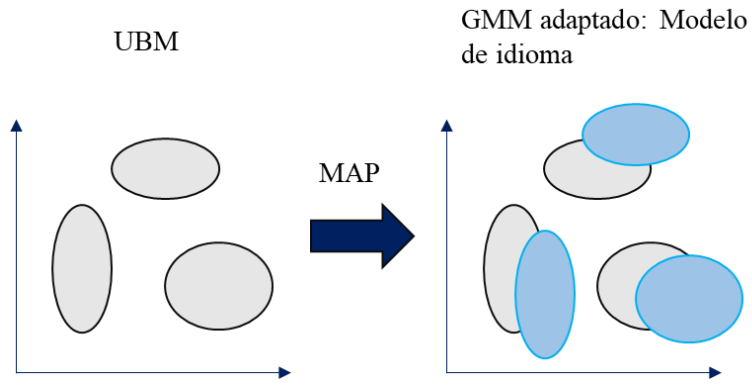


Figura 2-3: Adaptación GMM-UBM a través del algoritmo Máximo a Posteriori (MAP).

A partir del enfoque anterior, surge el concepto de *supervector*, el cual consiste en la concatenación de las medias de las componentes Gaussianas para cada idioma. Un *supervector* intenta representar de forma estable las características de un idioma independientemente de la longitud del segmento de voz. Idealmente se espera que cada segmento que contiene un mismo idioma devuelva el mismo *supervector* de dimensión fija. Este concepto puede ser interpretado como un mapeo entre una secuencia de habla y un vector de alta dimensión con un tamaño fijo [11].

2.1.5 iVectors

Siguiendo los avances logrados en los sistemas de reconocimiento de locutor, el uso de *iVectors* (provenientes del modelado de variabilidad total), seguido por diversos

mecanismos de clasificación, se han convertido en el estado del arte en los sistemas LID [12].

La extracción de *iVectors* [13] proporciona una forma de obtener una representación de tamaño fijo en un espacio de bajas dimensiones, condensando la información de variabilidad de los datos, y preservando la información específica del idioma. La idea general es que a partir de un UBM se aprenda un subespacio de dimensiones inferiores de una gran cantidad de datos y posteriormente se proyecte un segmento de habla en este subespacio, donde el vector de coordenadas es denominado *iVector*. Este enfoque surge a partir del éxito en la aplicación de la técnica *Join Factor Analysis* (JFA), la cual modelaba por separado la variabilidad del hablante e inter-sesión para intentar suprimir esta última, pero ciertos experimentos llevados a cabo demostraron que esta supresión también eliminaba información del hablante [14] [15]. A diferencia de la técnica JFA, la extracción de *iVectors* recoge toda la información de variabilidad importante en un mismo espacio de bajas dimensiones.

De esta forma, se logra solventar la principal desventaja de los sistemas GMM-UBM, la cual era un alto coste computacional debido a la alta dimensionalidad de los *supervectores* obtenidos.

2.2 Redes Neuronales

Durante años, el ser humano ha intentado emular el funcionamiento de nuestro cerebro, el cual se diferencia de forma notable del funcionamiento de una computadora convencional. El cerebro humano es un sistema altamente complejo, no linear y que procesa información de forma paralela. Tiene la capacidad de organizar los componentes de su estructura, denominados neuronas, para realizar ciertas operaciones de cálculo muchas veces más rápido que las computadoras más rápidas que existen.

Las redes neuronales son una serie de algoritmos que están basados en el funcionamiento de nuestro cerebro con el objetivo de reconocer patrones y poder realizar operaciones de cálculo altamente complejas. Más formalmente, en [16] se define más en profundidad el concepto, el cual es citado a continuación:

"Una red neuronal es un procesador distribuido masivamente paralelo formado por simples unidades de procesamiento que tiene una propensión natural para almacenar conocimiento experiencial y hacerlo disponible para su uso. Se parece al cerebro en dos aspectos:

1. El conocimiento es adquirido por la red desde su entorno a través de un proceso de aprendizaje.
2. Los pesos de las conexiones entre las neuronas, conocidos como pesos sinápticos, se utilizan para almacenar el conocimiento adquirido."

En la Figura 2-4, se puede observar la representación de una estructura básica de una red neuronal, la cual recibe una entrada ponderada y calcula una salida a través de una función de activación (normalmente una transformación no lineal), de la cual se hablará más adelante.

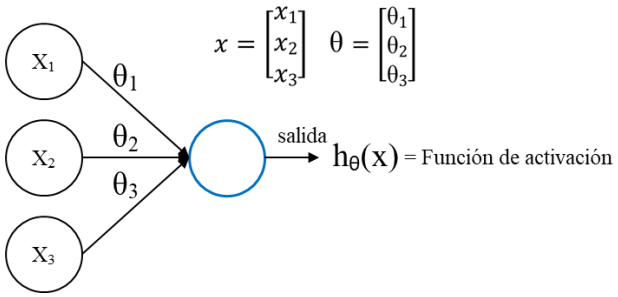


Figura 2-4: Modelo neuronal: estructura básica.

Combinando múltiples estructuras como las mostradas anteriormente, se puede crear una red neuronal sencilla. La Figura 2-5 muestra un ejemplo de una red neuronal con una capa de entrada, una oculta y una de salida.

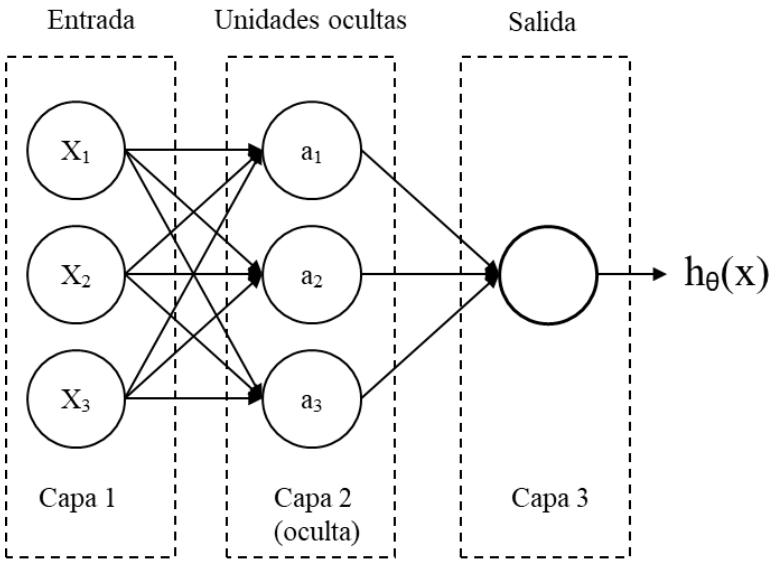


Figura 2-5: Estructura de una red neuronal sencilla.

En una red neuronal, a las capas intermedias situadas entre la entrada y la salida se les denomina capas ocultas. En el ejemplo anterior, las unidades que componen la capa oculta son llamadas unidades ocultas. Cada una de estas unidades intermedias es el resultado de aplicar una función de activación a la suma ponderada de cada elemento de la entrada (cada salida de las unidades de la capa anterior, conectada mediante los pesos a dicha unidad). De esta misma forma se calcula la salida, solo que esta vez tendrá el resultado de las unidades ocultas en vez de la entrada para realizar las operaciones. A continuación, se muestran las ecuaciones correspondientes al proceso descrito anteriormente:

$$a_1 = g(\theta_{11}^1 x_1 + \theta_{12}^1 x_2 + \theta_{13}^1 x_3)$$

$$a_2 = g(\theta_{21}^1 x_1 + \theta_{22}^1 x_2 + \theta_{23}^1 x_3)$$

$$a_3 = g(\theta_{31}^1 x_1 + \theta_{32}^1 x_2 + \theta_{33}^1 x_3)$$

$$h_\theta(x) = g(\theta_{11}^2 a_1 + \theta_{12}^2 a_2 + \theta_{13}^2 a_3)$$

Donde:

a_i: *unidades ocultas*
x_i: *entrada*
$g(\theta x)$: *función de activación*
θ_j^i: *pesos (i: número de la capa oculta)*
$h_\theta(x)$: *salida (hipótesis)*

Cómo se puede intuir, existen innumerables combinaciones a la hora de crear una red neuronal. Más adelante se verán algunos ejemplos de las arquitecturas más utilizadas.

2.2.1 Conceptos básicos: Función de activación, Forward propagation, función de coste y Backpropagation

La función de activación [17] es la transformación que aplica una unidad oculta, y que determina el resultado que generará una unidad de la red sobre su entrada, permitiendo que la red neuronal sea capaz de aprender transformaciones complejas de las entradas para una tarea en concreto (para la que se entrena).

Algunas de las funciones de activación más utilizadas son *Sigmoid*, *Tanh* y *ReLu*.

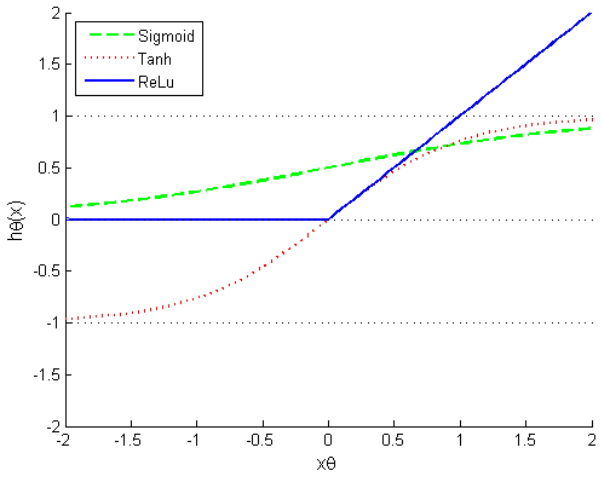


Figura 2-6: Ejemplos de funciones de activación.

Generalmente la función *Sigmoid* se utiliza en la capa de salida en problemas de clasificación binaria, debido a que devuelve valores entre 0 o 1, correspondientes a la probabilidad de que para la entrada del sistema se clasifique una de las dos clases. La función *Tanh* devuelve valores entre 1 y -1, rango que tiene media cero y resulta de utilidad para los cálculos de la red neuronal ya que normalmente los datos de entrada tienen también media cero. La función *ReLu* por su parte hace que los cálculos de la red sean más rápidos y se ahorre tiempo durante el entrenamiento [18]. El rendimiento del sistema en función de la función de activación que elijamos depende de numerosos factores, como pueden ser la función de coste utilizada, la estructura de la red, la media que elijamos para evaluar el rendimiento del sistema y la capa en la que se aplique cada tipo de función entre otros.

El funcionamiento básico de una red neuronal está basado en estos tres elementos: *Forward propagation*, Función de coste y *Backpropagation*. A continuación, se pasará a describir de forma breve en qué consisten cada uno de estos elementos, sin entrar a analizar sus correspondientes ecuaciones.

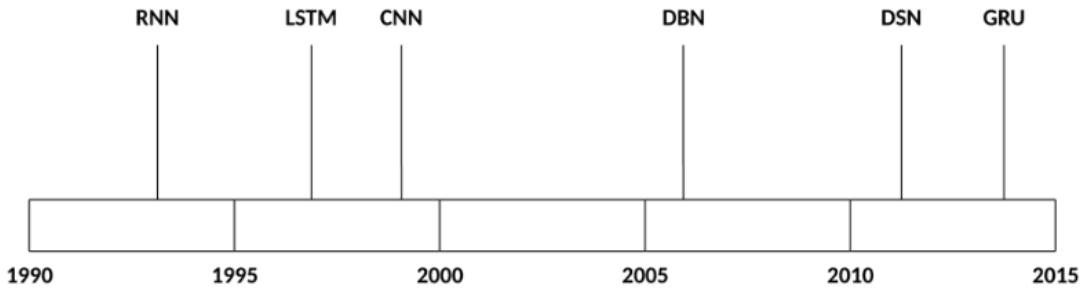
En una red neuronal, el término *Forward propagation* se refiere a las operaciones del conjunto de ecuaciones que permiten realizar los cálculos necesarios en todos los nodos de la red para obtener una salida a partir de una entrada. Si se vuelve a observar la Figura 2-5, se corresponde con el flujo de cálculos que se llevarían a cabo desde la parte izquierda de la red a la parte derecha.

La función de coste es la función objetivo que se pretende minimizar durante el entrenamiento de la red. A grandes rasgos, es la encargada de medir la diferencia entre la salida real de nuestra red en un momento dado y la salida deseada. Existe una gran variedad de funciones de coste que podemos utilizar cuando se diseña una red neuronal. Una de las más utilizadas es el error cuadrático medio (*Mean Squared Error*, MSE), la cual indica qué tan cerca está una hipótesis de un conjunto de puntos. Lo hace tomando las distancias de los puntos a la hipótesis y elevando al cuadrado. Elevar al cuadrado es necesario para eliminar cualquier signo negativo. También da más peso a las diferencias más grandes. Otra de las funciones más utilizadas es la entropía cruzada, la cual mide el rendimiento de un modelo de clasificación cuyo resultado es un valor de probabilidad entre 0 y 1. El valor de esta función aumenta a medida que la probabilidad predicha por nuestro sistema diverge de la etiqueta real.

A la hora de entrenar una red neuronal, como en cualquier problema de optimización, uno de los objetivos que se busca es minimizar la función de coste. Este proceso se lleva a cabo a través del cálculo de los gradientes junto con el algoritmo *Backpropagation* [19]. Este último consiste en el procedimiento de ajustar repetidamente los pesos de las conexiones de la red. Cuando se entrena una red neuronal [20], se piensa en la función de coste como una función de los parámetros (números describiendo cómo se comporta una red). Al calcular las derivadas de la función de coste con respecto a todos los parámetros, se encuentra que normalmente existen millones de parámetros en una red neuronal. El algoritmo *Backpropagation* puede ser interpretado como una herramienta que permite propagar y aproximar las derivadas a lo largo de la red en base a los parámetros de esta. Luego este algoritmo es un elemento fundamental de la red, ya que hace posible y tratable el entrenamiento de modelos más profundos y complejos. De forma intuitiva, este algoritmo representa el flujo del error de una red desde la salida a la entrada

2.2.2 Arquitecturas

Como se ha mencionado anteriormente en la sección 2.2, se pueden crear innumerables modelos en una red. En este apartado se mencionarán algunos ejemplos de las más utilizadas y los campos donde generalmente se aplican. Para ello se basará en [21].



RNN: *Recurrent Neural Networks*
LSTM: *Long short-term memory*
CNN: *Convolutional Neural Networks*
DBN: *Deep belief networks*
DSN: *Deep stacking networks*
GRU: *Gated Recurrent Unit*

Figura 2-7: Ejemplos de arquitecturas de redes neuronales.
*Figura extraída de [21]***.**

Tabla 1: Ejemplo de arquitecturas de redes neuronales más utilizadas y sus campos de aplicación. Tabla extraída de [21].

Arquitectura	***Aplicaciones***
RNN	Reconocimiento de habla y escritura.
LSTM/GRU	Comprensión del lenguaje natural, subtítulos de imágenes, reconocimiento de habla, escritura y gestos.
CNN	Reconocimiento de imagen, análisis de video, procesamiento del lenguaje natural (*Natural Language Processing*, NPL).
DBN	Reconocimiento de imagen, retención de información, entendimiento del lenguaje natural, predicción de fallos.
DSN	Retención de información, reconocimiento de habla continuo.

2.2.2.1 Redes Neuronales Recurrentes (RNNs)

Si se analiza el proceso de aprendizaje del ser humano, se puede encontrar que en numerosas ocasiones aprendemos algo nuevo basándonos en algo que ya sabíamos de antes. Por ejemplo, a medida que vamos leyendo este párrafo, nosotros vamos entendiendo cada palabra en gran medida basándonos en palabras previas (contexto). De esta forma, nosotros no nos olvidamos de lo que hemos leído anteriormente y empezamos desde cero. Las redes neuronales tradicionales no son capaces de realizar este procesamiento basado en

lo visto anteriormente, es por ello por lo que surgieron las Redes Neuronales Recurrentes (*Recurrent Neural Network,* RNNs).

Una Red Neuronal Recurrente (RNN) [22], es un modelo o tipo de arquitectura de red neuronal creado para modelar series temporales. La estructura de la red es similar a una red estándar de varias capas, con la diferencia de que permite conexiones a las unidades ocultas que están asociadas con un retardo en el tiempo. A través de estas conexiones, el modelo es capaz de retener información del pasado, permitiendo descubrir correlaciones entre eventos que están muy separados los unos de los otros en el tiempo. En la Figura 2-8 se muestra la estructura básica de una RNN.

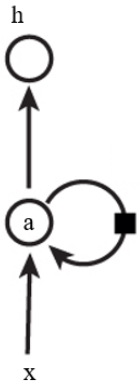


Figura 2-8: Estructura de una RNN.

Una RNN, puede ser vista como una serie de múltiples copias de la misma red, cada uno pasando un valor a un sucesor. Si "desenrollamos" una RNN, el resultado puede ser visto en la Figura 2-9.

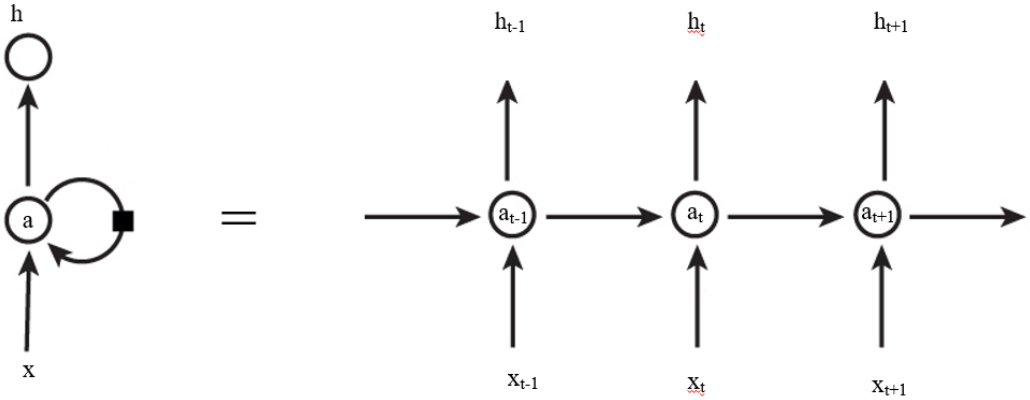


Figura 2-9: Resultado de "desenrollar" una RNN.

A pesar de que en principio el modelo de una RNN parece simple y potente, en la práctica es muy difícil entrenar estas redes y conseguir los resultados deseados. Los principales motivos de lo anterior son los problemas conocidos como *vanishing gradient* y *exploding gradient.*

El problema *exploding gradient* [23] se refiere al incontrolado incremento que pueden tener los gradientes durante el entrenamiento. En una RNN muy profunda, ciertos componentes pueden crecer exponencialmente resultando en un fallo del sistema. Por el lado contrario, el problema *vanishing gradient* se refiere al comportamiento opuesto,

donde ciertos componentes pueden decrecer muy rápido hacia el valor cero, haciendo imposible la tarea de aprender correlaciones entre eventos muy distanciados en el tiempo.

Dependiendo de la función de activación que utilicemos, se podrá dar un problema u otro. El problema *exploding gradient* se puede identificar y solventar fácilmente cuando ocurre, ya que los gradientes tomarán valor NaN y el programa fallará. Una solución sencilla y eficaz es establecer un umbral que marque el valor máximo que puede tomar una variable. Por otro lado, el problema *vanishing gradient* no queda tan evidente cuando ocurre ni tampoco como se debe tratar con él.

2.2.2.2 Long short-term memory (LSTM)

Las redes *Long short-term memory* (LSTM), son un tipo de RNNs creadas para dar solución al problema *vanishing gradient*. Fueron introducidas por primera vez en [23] y desde entonces han sido mejoradas y popularizadas. Hoy en día son utilizadas ampliamente en una gran variedad de problemas, especialmente para el modelado de secuencias temporales.

Una capa LSTM [24] consiste en un conjunto de bloques conectados de forma recurrente, conocidos como bloques de memoria. Estos bloques pueden ser interpretados como una versión diferenciable de los chips de memoria en una computadora. Por esta razón, las unidades de una LSTM algunas veces se les llama celdas de memoria.

En este trabajo no se entrará a ver en detalle el funcionamiento de una celda de memoria de una LSTM. Por lo tanto y para nuestro objetivo, las celdas pueden ser vistas como una unidad que toma como entrada el estado previo y la entrada actual. Internamente estas celdas deciden qué información mantienen en la memoria u olvidan. Después combinan el estado previo, la entrada actual y la información en memoria actual. Así, este tipo de unidades son muy eficientes a la hora de hallar correlaciones entre eventos muy lejanos en el tiempo. Información detallada del funcionamiento de una celda LSTM puede ser encontrada en [25]. En la Figura 2-9 se puede observar un ejemplo de una celda de una LSTM.

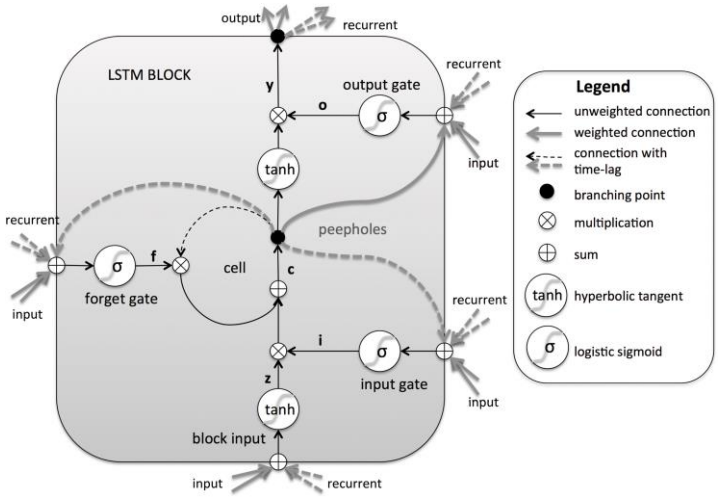


Figura 2-10: Ejemplo de una celda de memoria de una LSTM.
Figura extraída de [4].

2.2.2.3 Bottleneck Neural-Network (BN-NN)

Hasta ahora solo se ha tenido principalmente en cuenta a los MFFC como características de entrada de una red neuronal para tareas de reconocimiento de idioma.

Sin embargo, una red neuronal también se puede utilizar como un medio de extracción de características para ser tomadas como entrada de otra red neuronal. Se denomina *Bottleneck Neural-Network* (BN-NN) [26] a la arquitectura de la primera red neuronal (*feed-forward*), en donde una de las capas ocultas tiene significativamente menor dimensión que las capas de su alrededor. Se asume que dicha capa, llamada cuello de botella (*bottleneck*), comprime la información necesaria para mapear la entrada a la salida, aumentando la solidez del sistema al ruido y al sobreajuste. El vector de salida de esta capa de menor dimensionalidad es denominado como características *bottlenecks*.

Un vector de características *bottlenecks* [26] de un segmento de audio puede ser interpretado como una comprensión de la información fonética presente en un vector de menor dimensión (la red se entrena para clasificación de unidades fonéticas). En el sistema propuesto en este trabajo, veremos cómo se utilizan estas características y el impacto que tiene en el rendimiento.

2.3 Sistemas de reconocimiento de idiomas basados en Redes Neuronales.

Enfoques basados exclusivamente en aplicar redes neuronales tomando como entradas características como los MFCC muestran un igual o mejor rendimiento que las técnicas LID del estado del arte (*iVectors*), a la vez que son menos complejos. Además de estos enfoques, también se pueden encontrar redes utilizadas como extractores de características, otras *end-to-end* y recientemente otras utilizadas como extractores "*embeddings*" A continuación, se verán algunos ejemplos en los que las redes neuronales han sido empleadas con éxito en la identificación de idiomas. Para más información de cada uno de estos sistemas se debe consultar su respectiva referencia.

En [4] se utilizan coeficientes cepstrales en la escala de Mel con coeficientes delta desplazados (MFCC-SDC) como características de entrada a una red LSTM unidireccional, cuya salida se usa para la clasificación de idioma. La última predicción del clasificador contiene a el idioma predicho. Con este sistema se logra mejorar el rendimiento de un sistema *iVector* en un 26% teniendo como referencia la medida *Equal Error Rate*. Este trabajo de fin de grado está basado parcialmente en este artículo de investigación. Otros trabajos basados en ideas similares son el empleo de DNNs en lugar de las LSTMs. En [12] y en [27] se implementan también múltiples arquitecturas de una DNN, consiguiendo un rendimiento superior al de un sistema *iVector* tradicional.

En [28] se propone un sistema *end-to-end* basado en CNNs, en donde transforman los datos de entrada en una imagen que contiene características MFCC-SDC. El eje x de esa imagen representa el dominio del tiempo y el eje y se corresponde con los valores de los coeficientes. Este sistema, combinado con el sistema base de referencia *iVector*, logra una mejora relativa del 11%. Otro sistema similar al anterior se propone en [3], donde se implementa una combinación de una CNN y una RNN con el objetivo de aprovechar la

habilidad descriptiva de la primera red con la habilidad de capturar patrones temporales de la segunda red. En este caso también se emplean MFCC-SDC como características de entrada.

En [29] se utiliza una red bidireccional LSTM para capturar información del lenguaje, tomando como entrada coeficientes *Perceptual Linear Prediction* (PLP) y su primera y segunda derivada. Para obtener un solo vector de clasificación, se calcula la media geométrica de todos los vectores de salida en las secuencias de salida (*forward y backward*).

3 Tecnologías a utilizar

Para la realización del diseño y desarrollo de este proyecto se ha utilizado el lenguaje de programación Python y diversas librerías que cuentan con funcionalidades que resultan especialmente útiles en proyectos de aprendizaje automático.

Las razones principales por las que se ha decidido utilizar Python son las siguientes:

- Relación complejidad/rendimiento: Gracias a la simplicidad de su sintaxis, es un lenguaje fácil de implementar en comparación con otros con mejor rendimiento como C o C++, los cuales requieren una cantidad de tiempo considerablemente mayor a la hora de desarrollar y corregir el código. Además, Python presenta en muchas ocasiones un mayor rendimiento en comparación con otros lenguajes como R o Matlab.

- Gran número de librerías: Cuenta con numerosas librerías orientadas al aprendizaje automático cuyo código se encuentra en repositorios de código abierto que son desarrollados por personas para mejorar continuamente las funcionalidades existentes.

- Amplio soporte: Como consecuencia de los dos puntos anteriores, Python se está convirtiendo en el lenguaje de referencia para proyectos de aprendizaje automático. Gracias al aumento de su popularidad, se puede encontrar en internet un amplio soporte por parte de la comunidad.

A continuación, se mencionarán las principales herramientas, librerías, módulos y paquetes utilizados en el proyecto.

3.1 Keras

Keras [30] es una librería de alto nivel para trabajar con redes neuronales, está escrita en Python y es capaz de ejecutarse sobre TensorFlow, CNTK o Theano. Fue desarrollada con un enfoque que permite la rápida experimentación, siendo capaz de pasar de la idea al resultado con el menor tiempo posible, lo que resulta fundamental en la realización de una buena investigación basada en resultados empíricos.

Los principales motivos por los que se ha elegido Keras son los siguientes:

- Fácil implementación: Permite una rápida y simple implementación gracias a su modularidad, extensibilidad y su amigable sintaxis para el usuario.

- Soporta LSTMs: Incluye funcionalidades para implementar redes LSTMs.

- Posibilidad de ejecución en unidades de procesamiento gráfico (*Graphics Processing Unit*, GPUs): Permite ejecutar los programas sobre GPUs, punto que resulta clave en la ejecución de este proyecto.

La librería de Keras se utiliza en este trabajo para la creación de los múltiples modelos, el entrenamiento y la evaluación de los diferentes tipos de redes neuronales empleadas.

3.2 TensorFlow

TensorFlow [31] es una interfaz para expresar los algoritmos de aprendizaje automático y una implementación para ejecutar tales algoritmos. Un cálculo expresado usando TensorFlow puede ser ejecutado con poco o ningún cambio en una amplia variedad de heterogéneos sistemas, que van desde dispositivos móviles como teléfonos y tabletas hasta sistemas distribuidos a gran escala de cientos de máquinas y miles de dispositivos computacionales como Tarjetas GPU.

TensorFlow se ha elegido en este proyecto como *backend* de Keras. Las razones principales han sido las siguientes:

- A diferencia de Theano, TensorFlow es capaz de descubrir automáticamente todas las GPUs que hay disponibles en el sistema y ejecutarse en estas aprovechando su máxima capacidad.

- En comparación con CNTK, cuenta con un amplio soporte por parte de la comunidad.

Por lo tanto, al ser backend de Keras, TensorFlow interviene al igual que Keras en la creación de los múltiples modelos, el entrenamiento y la evaluación de los diferentes tipos de redes neuronales empleadas.

3.3 Otras librerías

En esta sección se listarán y describirán de forma breve otras librerías que han sido utilizadas.

NumPy: Es una librería que permite principalmente trabajar con matrices multidimensionales, de forma que las operaciones matemáticas realizadas sobre estas tengan un rendimiento elevado.

scikit-learn: Es un paquete que proporciona diferentes algoritmos de aprendizaje automático. En este proyecto se ha utilizado para la evaluación del rendimiento de los sistemas mediante el cómputo de las matrices de confusión.

LibROSA: Es una librería que proporciona diferentes funcionalidades enfocadas al análisis de la señal de audio. Se ha empleado en el proyecto para el cálculo de las características deltas, que equivalen a las derivadas de los datos de entrada de la red a lo largo de una dimensión.

4 Diseño

4.1 Base de datos utilizada

En el desarrollo de este trabajo, se ha utilizado la base de datos proporcionada por el *National Institute of Standards and Technology* (NIST) *Language Recognition Evaluation* (LRE) en 2009, cuyo objetivo principal es la proporción de un marco de trabajo que permite evaluar y comparar el rendimiento de los sistemas relacionados con la tarea del reconocimiento del idioma en segmentos extraídos de conversaciones telefónicas.

La base de datos de LRE 2009, incluye datos divididos en dos categorías:

- Datos de transmisiones de noticias provenientes de "*Voice of America*" (VOA), donde están mezclados segmentos de habla telefónica y no telefónica.

- Conversaciones telefónicas (*Conversational Telephone Speech*, CTS), consistentes en conversaciones telefónicas espontáneas.

Ambas categorías contienen datos en crudo muestreados a 8KHz.

Al igual que en [4] y en [12], debido a la variedad del número de horas disponibles para cada idioma, se han escogido 8 idiomas a evaluar de los 40 con los que contaba la base de datos. Estos 8 idiomas escogidos cuentan con más de 200 horas de datos. Además, para evitar interpretaciones que puedan dar lugar a conclusiones incorrectas o dudosas debido a la diferente naturaleza de los datos de cada categoría, para el entrenamiento, la validación y el conjunto de test de los modelos propuestos se han empleados datos exclusivamente de VOA.

Los 8 idiomas seleccionados son Chino Mandarín (zh), Dari (fa), Inglés estadounidense (en), Francés (fr), Pashto (ps), Ruso (ru), Español (es) y Urdu (ur).

4.2 Medidas de evaluación empleadas y sistema de referencia

Para evaluar el rendimiento del sistema propuesto, se ha utilizado las tres siguientes medidas:

- *Accuracy:* Indica el porcentaje de predicciones correctas que ha realizado nuestro sistema.
- *Cavg* (media del coste): Está definida en [32] y mide el coste de tomar decisiones incorrectas.
- *Matriz de confusión:* Se utiliza como medida auxiliar con el objetivo de analizar más en profundidad el rendimiento final del mejor sistema para cada idioma. Esta matriz contiene información de las clasificaciones predichas y las objetivo, de forma que permite identificar fácilmente donde se está equivocando el sistema para cada clase.

Con el objetivo de comparar adicionalmente el modelo, se ha utilizado como referencia un sistema *iVector* descrito en [4].

4.3 Metodología utilizada

En este apartado se pretende describir el flujo del modelo de pruebas que hemos utilizado. Antes de empezar con la descripción es conveniente indicar que, en cada uno de los experimentos llevados a cabo el entrenamiento de las redes neuronales implementadas en este trabajo conlleva días de duración, lo que condiciona en cierta medida el número de experimentos y algunos de los parámetros elegidos en el diseño en cada uno de ellos.

La idea seguida a la hora de realizar los experimentos ha sido empezar con un modelo más simple y posteriormente ir incrementando la complejidad del sistema para poder observar la variación del rendimiento de la red en función de los múltiples hiperparámetros, los cuales son variables establecidas antes de optimizar realmente los parámetros del modelo (pesos) y que están relacionados con la arquitectura de la red y el proceso de entrenamiento entre otros elementos.

Además de ir aumentando la complejidad del sistema, en los experimentos se utilizaron como características de entrada MFCC, MFCC con sus respectivas derivadas de primer y segundo orden (MFCC + Δ + $\Delta\Delta$) y características *bottlenecks*.

4.4 Definición de hiperparámetros

En esta sección se van a analizar algunos de los principales hiperparámetros de la red. Antes se va a diferenciar entre los hiperparámetros que se han modificado a lo largo de la realización de los diferentes experimentos y los que se han mantenido de forma fija para todos los experimentos.

Para empezar, se muestran los hiperparámetros que se han mantenido de forma fija a lo largo de los experimentos y la justificación de por qué han sido elegidos.

Función de activación: Como ya se ha comentado en la sección 2.2.1 de esta memoria, la función de activación es la transformación que aplica una unidad oculta. En este trabajo se utilizan dos tipos de funciones de activación diferentes.

- *Sigmoid*: Se utiliza en todas las unidades de la red excepto en las de la capa de salida. El motivo por el que se ha elegido esta función es debido a que es una de las más utilizadas y no es de las más costosas computacionalmente.
- *Softmax*: Es la más utilizada en la capa de salida en los problemas de multi-clasificación. Esta función normaliza los valores entre 0 y 1, correspondientes a la probabilidad de que cada unidad corresponda a una clase y por lo tanto el valor del sumatorio de todas las probabilidades es igual a 1. En este trabajo se ha empleado la función *Softmax* en la capa de salida.

Función de coste: Como ya se ha mencionado en este trabajo, el objetivo del entrenamiento de una red neuronal es minimizar la función de coste. En los experimentos se ha utilizado la entropía cruzada. Los motivos por la que se ha elegido son debido a que

es una de las más utilizadas en los problemas de multi-clasificación y en combinación con la función de activación *Sigmoid* ayuda a que el proceso de aprendizaje sea más rápido y a enfrentar el problema de *vanishing gradient*. No se va a entrar en detalle del razonamiento detrás de estos dos últimos motivos debido a su complejidad y extensión, pero básicamente y de forma resumida, consiste en que a la hora de calcular los gradientes en una red neuronal y aplicar el algoritmo de *backpropagation* junto con la entropía cruzada, en las ecuaciones se excluye el término que deriva a la función de activación. Si miramos la función *Sigmoid* en la Figura 2-6, vemos que si toma valores próximos a los extremos la derivada resulta en un valor muy bajo, y por lo tanto propicia la aparición del problema del *vanishing gradient.* La justificación de los motivos anteriores puede ser encontrada en detalle en [33].

***Dropout*:** Es una técnica empleada para evitar el sobre-entrenamiento. La idea consiste en ir descartando aleatoriamente unidades de la red (junto con sus pesos) durante el entrenamiento. De esta forma se evita que las unidades establezcan dependencias entre sí. Esta técnica puede ser empleada en cualquiera de las capas de la red neuronal, sin embargo, en este trabajo se ha optado por aplicarla únicamente en primera capa oculta, con un valor de 0.3, que es el equivalente a descartar un 30% de las unidades ocultas. Un ejemplo del efecto que tiene la aplicación de esta técnica sobre la estructura de la red se muestra en la Figura 4-1.

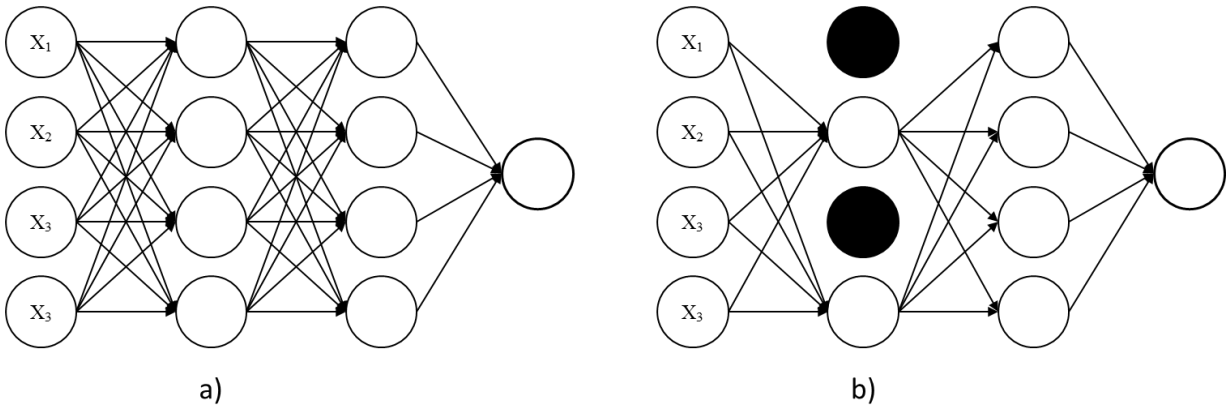


Figura 4-1: a) Antes de aplicar *dropout y* b) Después de aplicar *droput* con un valor de 0.5 en la primera capa oculta.

Factor de aprendizaje: El factor de aprendizaje es el parámetro que controla cuánto estamos ajustando los pesos con respecto a los gradientes, es decir, cuánto de grande es el paso en la dirección opuesta al gradiente en cada actualización de los parámetros. Debido a que el valor de los gradientes puede sufrir grandes variaciones para los diferentes pesos de la red durante el entrenamiento, es muy complejo elegir un único valor del factor de aprendizaje. Es por ello por lo que se ha empleado el optimizador *RMSprop*, el cual consiste en un método para adaptar el factor de aprendizaje durante el entrenamiento. Es el optimizador recomendado en la documentación técnica de Keras cuando trabajamos con RNNs.

Tamaño del *mini-batch*: Indica el número de muestras que evalúa la red entre cada actualización de los pesos de esta. Teóricamente un valor mayor equivale a una mayor

eficiencia en la computación de cálculos debido a la utilización de arquitecturas de computación paralela. Sin embargo, un valor demasiado grande puede llevar a la obtención de pesos no deseados para el rendimiento del sistema. El valor elegido para nuestra red es 200.

Una vez vistos los parámetros que hemos mantenido de forma fija, se pasarán a analizar los que se han ido modificando entre los diferentes experimentos y la justificación de por qué se han elegido.

Número de unidades por capa oculta: Debido a que un valor mayor de este parámetro supone un mayor coste computacional y a que en [4] estos dos valores fueros utilizados obteniendo resultados satisfactorios, hemos optado por utilizar 256 unidades por cada capa oculta y posteriormente 512 unidades.

Tipo de Capa: Cuando se intenta reconocer un idioma, resulta útil conocer información que se ha escuchado previamente en el mismo segmento de voz, no nos basamos solo en una trama de audio. Es por ello por lo que el tipo de capa principal que se ha utilizado es una LSTM, la cual es capaz de recordar información previamente vista, teniendo en cuenta y modelando de forma eficiente el contexto de la trama actual. También se ha empleado en algunos experimentos una capa *fully-connected* con el objetivo de ver el efecto que tiene sobre la red.

Dimensión de las características de entrada: Se ha variado el tipo de características de entrada, y por tanto, las dimensiones de su capa de entrada. Para los MFCC, se cuenta con 20 coeficientes por trama, para los MFCC con derivadas de primer y segundo orden se cuenta con 60 coeficientes en total (20 correspondientes a las primeras derivadas y 20 a las segundas derivadas, además de los 20 de los MFCC) y para el caso de los *bottlenecks* se tienen 80 coeficientes por trama.

Épocas: Este término indica el número de veces que el conjunto entero de datos ha sido expuesto en una red. Teóricamente y de forma general, a medida que se aumenta el número de épocas aumenta el rendimiento de nuestro sistema, hasta que alcanza el punto máximo de mejora (óptimo de convergencia del algoritmo). Para los experimentos se ha seleccionado el número de épocas en función del incremento del *accuracy* correspondiente al conjunto de validación. Si el *accuracy* deja de incrementarse durante un período de tiempo previamente determinado entonces se detiene el entrenamiento del sistema.

4.5 Experimentos llevados a cabo

En total se llevaron a cabo cinco experimentos siguiendo la metodología descrita en la sección 4.3. Para distinguir cada uno de los experimentos, se les llamarán respectivamente *Exp_1, Exp_2, Exp_3, Exp_4* y *Exp_5.*

Exp_1: Se empleó una red LSTM de 2 capas ocultas con 256 unidades en cada capa. Las características de entrada empleadas fueron 20 MFCC por cada trama de audio.

Exp_2: Se utilizó una red LSTM de 2 capas ocultas con 512 unidades en cada capa. Las características de entrada empleadas fueron 20 MFCC por cada trama de audio.

Exp_3: Se empleó una red neuronal de 3 capas ocultas con 512 unidades en cada capa. Las dos primeras con una arquitectura LSTM y la última capa con una arquitectura *fully-connected*. Las características de entrada empleadas fueron 20 MFCC por cada trama de audio.

Exp_4: La arquitectura del modelo es igual a la del *Exp_3*, con la diferencia de que se añadieron a las características de entrada su primera y segunda derivada, quedando en 20 MFCC, 20 derivadas de primer orden y 20 derivadas de segundo orden, haciendo un total de 60 valores para cada trama.

Exp_5: La arquitectura del modelo es igual a la del *Exp_3 y Exp_4*, con la diferencia de que se utilizaron como entrada 80 valores por trama correspondientes a las características *bottlenecks.*

En la Tabla 2 se muestra un resumen con las especificaciones que se fueron modificando de los diferentes experimentos llevados a cabo.

Tabla 2: *Resumen de las especificaciones de los experimentos llevados a cabo.*

Experimento	***Características de entrada***	***Capas ocultas***	***Unidades por capa oculta***	***Arquitectura de las capas***
Exp_1	MFCC	2	256	LSTM
Exp_2	MFCC	2	512	LSTM
Exp_3	MFCC	3	512	2 primeras capas LSTM y 1 *fully-connected*
Exp_4	MFCC + Δ + $\Delta\Delta$	3	512	2 primeras capas LSTM y 1 *fully-connected*
Exp_5	*bottlenecks*	3	512	2 primeras capas LSTM y 1 *fully-connected*

5 Desarrollo

En esta sección se describe el proceso de implementación de los experimentos descritos previamente. Para cada uno de los experimentos realizados se han llevado a cabo una serie de pasos en un orden determinado. Con el objetivo de lograr un mejor entendimiento del proceso realizado, se han dividido los pasos a través de dos módulos diferentes:

- Preprocesamiento de los datos
- Definición, entrenamiento y evaluación del modelo.

5.1 Preprocesamiento de los datos

Este módulo es de vital importancia ya que transforma la señal de audio en una serie de vectores de características sobre los que aprenderá nuestro sistema. Es por ello por lo que es fundamental un preprocesamiento adecuado de los datos.

Como ya se ha comentado previamente, se ha utilizado la base de datos proporcionada por el NIST LRE en 2009. En este trabajo cabe mencionar que el proceso de extracción de MFCC y de *bottlenecks* fue proporcionado con antelación. De esta forma, se empezó contando con todos los vectores correspondientes a cada trama de todos los segmentos de audio disponibles para cada uno de los ocho idiomas. Adicionalmente, resulta útil conocer los siguientes valores relacionados con el proceso de extracción de características:

- Los vectores fueron extraídos de ventanas de 20ms de duración.
- El solapamiento utilizado fue de 10ms.
- Para el caso de los MFCC, se cuenta con 20 coeficientes por cada trama.
- Para el caso de los *bottlenecks*, los cuales han sido extraídos de un elaborado proceso que parte de los MFCC y posteriormente pasan a través de una red entrenada para la tarea del reconocimiento automático del habla (*Automatic Speech Recognition*, ASR), se cuenta con 80 coeficientes por cada trama.

Este módulo está divido además en cuatro subsecciones para lograr una mejor estructuración del proceso.

5.1.1 Creación de los conjuntos de datos

Partiendo del conjunto total de datos seleccionados de NIST LRE 2009, se hizo una división en tres conjuntos de datos diferentes: conjunto de entrenamiento, de validación y de test. Esta división es necesaria para poder evaluar el rendimiento de nuestro sistema de forma correcta. El proceso de creación de cada uno de estos conjuntos es el mismo para el caso de los experimentos con MFCC y *bottlenecks*.

a) Conjunto de entrenamiento:

La creación del conjunto de entrenamiento, que como bien indica su nombre son los datos que fueron utilizados exclusivamente durante el entrenamiento y por lo tanto se

corresponden con los datos a los que mejor se adapta nuestra red y de los que aprende los parámetros. La evaluación del sistema basada únicamente en estos datos es errónea (y muy optimista), ya que son datos conocidos por el sistema y no representan a todos los datos del mundo real.

A continuación, se describe el proceso de creación de este conjunto.

De cada fichero de audio disponible, se han extraído de forma aleatoria secuencias de 10s de duración. Si la duración de los ficheros de audio es inferior a 10s, se han añadido ceros al final de la secuencia (*zero-padding*) hasta completar los 10s. En caso de que la duración del fichero sea superior a 10s, se ha cogido como punto inicial un momento aleatorio del fichero entre el rango [0s: duración total del fichero - 10s], asegurando de esta forma el coger siempre 10s. Una vez se dispone de las secuencias de audio de 10 segundos extraídas, se ha pasado a dividirlas en 3 secuencias de 3s de duración cada una, descartándose siempre el último segundo. Posteriormente se han mezclado todas las secuencias para obtener un conjunto de datos lo más aleatorio posible. Teniendo en cuenta los valores comentados previamente relacionados con el procesamiento de extracción de características, para cada secuencia final vamos a tener 300 tramas de 20 MFCC o 80 *bottlenecks* cada una. El conocimiento de este valor va a resultar útil posteriormente en la creación de nuestro modelo de red de neuronal. En total, el conjunto de entrenamiento cuenta con 200976 secuencias de 3s de duración cada una (aproximadamente 167 horas de voz).

b) Conjunto de validación:

Este conjunto se utiliza para evaluar el rendimiento del sistema en un conjunto de datos que la red no haya visto previamente durante el entrenamiento. De esta manera, el modelo seleccionado será el correspondiente a la época en la que se obtenga el mejor rendimiento (*accuracy*) en este conjunto. Idealmente, este conjunto debería representar las condiciones de los datos de test, aunque no siempre ocurre.

El proceso de creación de este conjunto es igual al proceso de creación del conjunto de entrenamiento, con la única diferencia de que la duración de cada secuencia extraída de los ficheros de audios es de 3s directamente, no teniéndose que dividir posteriormente en secuencias. En total, el conjunto de validación cuenta con 33496 secuencias de 3s de duración cada una (aproximadamente, 28 horas de voz).

c) Conjunto de test

Es el conjunto que se utiliza para evaluar el rendimiento final del sistema. Estos datos idealmente serían los datos del entorno de aplicación en un caso real. Con este conjunto, se logra obtener una medida de rendimiento cercana a lo que se debería obtener en un caso real de aplicación.

El proceso de creación es igual al proceso de creación del conjunto de validación. En total, el conjunto de test cuenta con 2942 secuencias de 3s de duración cada una (aproximadamente 2 horas y media de voz).

5.1.2 Verificación de la distribución de secuencias por idioma

Antes de proceder con cualquier otro paso, se debe comprobar que los conjuntos están balanceados en cuanto al número de secuencias por idioma. De forma contraria, no se consigue una medida del *accuracy* que represente de forma real el rendimiento del sistema.

Para entender el motivo anterior, imaginemos un caso ilustrativo. Supongamos que el español cuenta con el 99% del total del número de muestras en todos los conjuntos de datos. Dado que la red ha visto en su gran mayoría el idioma español durante el entrenamiento, en teoría aprenderá a reconocer de forma adecuada el español únicamente. Ahora si evaluamos el rendimiento del sistema en los conjuntos de validación y de test, que recordemos que también cuentan con 99% de secuencias correspondientes al español, la precisión de nuestro sistema resultará en un valor muy alto y, sin embargo, la red en realidad no sabe reconocer ningún otro idioma. Como se puede intuir, varias combinaciones en los valores de los porcentajes anteriores en los diferentes conjuntos de datos pueden provocar un valor del rendimiento erróneo.

En la Figura 5-1 se muestra la distribución de secuencias por idioma para cada conjunto de datos.

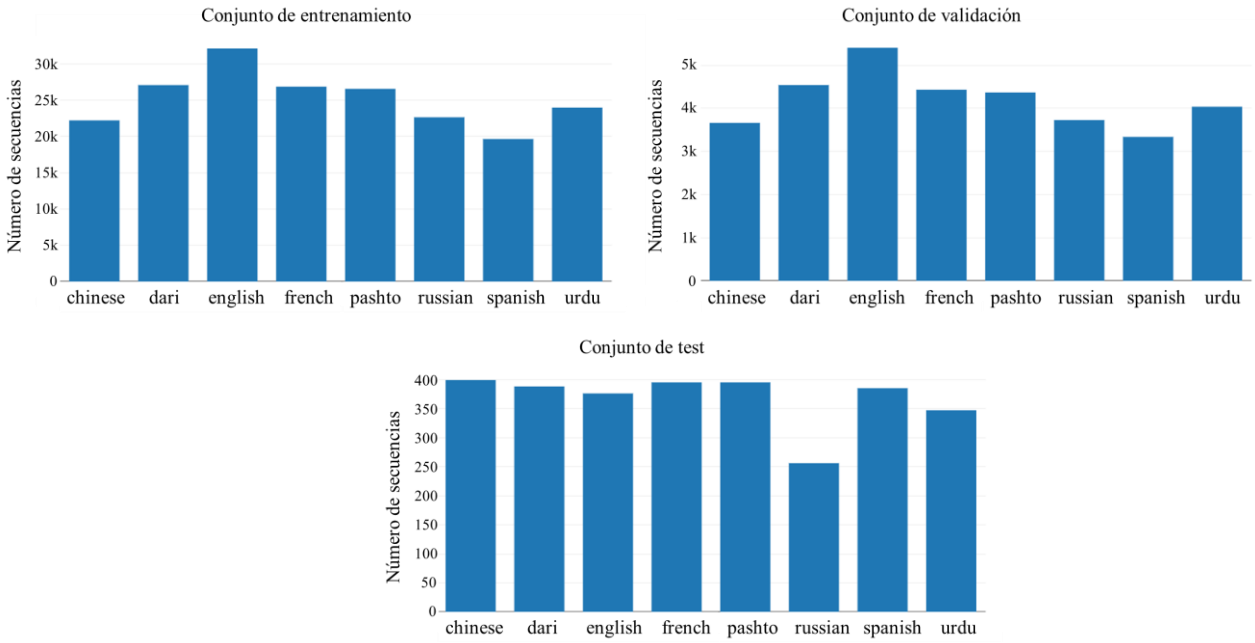


Figura 5-1: Distribución de secuencias por idioma para los conjuntos de entrenamiento, validación y test.

Como se puede observar, no hay grandes diferencias entre la distribución de los conjuntos ni una clase mayoritaria.

5.1.3 Cálculo de derivadas

Como se puede ver en la Tabla 2, en el *Exp_4* se han utilizado adicionalmente a los MFCC sus respectivas derivadas de primer y segundo orden, con el objetivo de proporcionar información adicional en la entrada de la red y poder ver el efecto que tienen sobre el rendimiento del sistema. Una vez calculadas las derivadas para cada conjunto de datos, se añadieron a continuación de los MFCC, resultando para cada trama en un vector de 60 valores donde los 20 primeros valores corresponden a los MFCC, los siguientes 20 valores a las derivadas de primer orden y los últimos 20 valores a las derivadas de segundo orden.

5.1.4 Normalización de los datos.

La normalización de los datos antes de empezar con el entrenamiento es un paso importante a la hora de implementar una red neuronal. Si las escalas son muy diferentes para los distintos valores de la entrada, los más grandes tendrán una mayor contribución al error de salida, por lo que la red se centrará en las variables de valores más altos, dando menos importancia la información que contiene valores más pequeños. Esto puede tener un notable impacto en el rendimiento de la red. Además, la no normalización de los datos de entrada también puede tener un impacto en el tiempo de cálculo necesario durante el entrenamiento.

En este trabajo se ha optado por una normalización global de los datos, con uno de los métodos más comunes de normalización de los datos:

$$x' = \frac{x - \bar{x}}{\sigma}$$

Donde:
x': valor normalizado
x: valor a normalizar
$\bar{x}$: media del conjunto de datos
σ: desviación típica del conjunto de datos

El proceso de normalización se debe de llevar a cabo en los conjuntos de datos de entrenamiento, validación y de test. Los valores de la media y la desviación típica son calculados a partir del conjunto de entrenamiento exclusivamente, y son estos valores los utilizados para normalizar los conjuntos de validación y test posteriormente.

5.2 Definición, entrenamiento y evaluación del modelo

Una vez se tienen los datos tratados de forma adecuada y listos para ser introducidos en la red, el siguiente paso es definir los modelos que se han diseñado en la sección 4 de este trabajo. Para ello se ha hecho uso principalmente de la librería de Keras.

Lo primero es definir el modelo de capas que se va a utilizar, de esta forma se indica la forma en las que las diferentes capas de la red van a estar organizadas. El modelo que se

utiliza comúnmente y el que se ha optado por utilizar en este trabajo es el *Secuencial*, el cual consiste en una agrupación linear de capas.

El siguiente paso es definir el proceso de aprendizaje, donde se indican los valores de los hiperparámetros que se han elegido y analizado previamente en el diseño para cada una de las capas definidas. Una de las grandes ventajas de la herramienta Keras es que cuenta con las principales funciones ya implementadas y optimizadas, de forma que pueden ser utilizadas con tan solo una llamada. Además, permite la flexibilidad de añadir funciones y algoritmos creados por nosotros mismos o por terceros, lo que permite mantener un desarrollo simple a la vez que el usuario final tiene el control absoluto cuando lo necesite.

Una vez se ha definido el modelo, el siguiente paso es pasar a entrenar la red neuronal con el conjunto de datos de entrenamiento.

A la hora de entrenar el modelo, un punto importante es la inicialización de los pesos de la red. Dependiendo de cómo se inicialicen, el rendimiento y resultado de la red puede sufrir variaciones. Una forma adecuada de inicializar los pesos es con valores aleatorios próximos a cero, ya que de esta forma se ayuda a romper la simetría de la red y cada neurona realiza un cálculo diferente, proporcionando un mejor valor del *accuracy.*

Como se menciona previamente en este trabajo, el proceso de entrenamiento de cada uno de los experimentos conlleva varios días de duración debido a la cantidad de datos y cálculos que maneja la red neuronal, incluso habiendo hecho uso de una GPU, la cual disminuye el tiempo requerido considerablemente en comparación con una unidad central de procesamiento (*Central Processing Unit,* CPU).

Durante el entrenamiento, se han ido guardando los datos necesarios para evaluar el sistema posteriormente. Además, para contrarrestar la posible aparición de fallos durante el proceso de entrenamiento y como consecuencia la pérdida del tiempo empleado teniendo que volver que empezar el proceso desde el principio, en cada época se guardó la red neuronal entera, incluyendo todos los pesos e hiperparámetros de la misma. De esta forma, ante cualquier fallo, la herramienta de Keras tiene la capacidad de reanudar el proceso entero desde el estado en el que se guardó.

Uno de los problemas encontrados durante el entrenamiento fue la imposibilidad de cargar todo el conjunto de datos de entrenamiento en memoria debido al tamaño de los datos y el límite de memoria. La solución que se ha llevado a cabo para solventar este problema ha sido dividir el conjunto de datos de entrenamiento en 82 subconjuntos e ir cargando en la memoria solo uno de ellos a la vez.

Por último, se ha evaluado el rendimiento del modelo a través tanto de las medidas que fueron definidas previamente (*accuracy*) en la red, como de las medidas que se decidan implementar posteriormente (C_{avg} y matriz de confusión).

6 Pruebas y resultados

En esta sección, se muestran y comparan los resultados de los diferentes experimentos realizados, teniendo en cuenta como medidas principales el *accuracy* y el C_{avg}. Además, se pretende dar una explicación justificada del rendimiento obtenido con cada sistema.

Cabe mencionar que en este apartado se muestran los principales gráficos e imágenes, el resto puede ser encontrado en el Anexo de este trabajo.

6.1 Resumen de resultados

En la Tabla 3 se muestra un resumen de los resultados obtenidos.

Tabla 3: *Resumen de los experimentos realizados y sus respectivos valores del rendimiento para 8 idiomas diferentes en secuencias de 3s.*

Experimento	***Características de entrada***	***Arquitectura de las capas***		***Rendimiento***	
ID			***Tamaño***	***Accuracy (%)***	C_{avg}
Exp_1	MFCC	LSTM (256)	~810k	37,12	0,35
Exp_2	MFCC	LSTM (512)	~3,2M	43,81	0,32
Exp_3	MFCC	2 primeras capas LSTM y 1 *fully-connected* (512)	~3,5M	45,62	0,31
Exp_4	MFCC + Δ + ΔΔ	2 primeras capas LSTM y 1 *fully-connected* (512)	~3,5M	41,94	0,32
Exp_5	*bottlenecks*	2 primeras capas LSTM y 1 *fully-connected* (512)	~3,6M	70,09	0,17

Por otro lado, en la Figura 6-1, se puede observar de forma más gráfica una comparación de las medidas del *accuracy* y el C_{avg} para cada experimento.

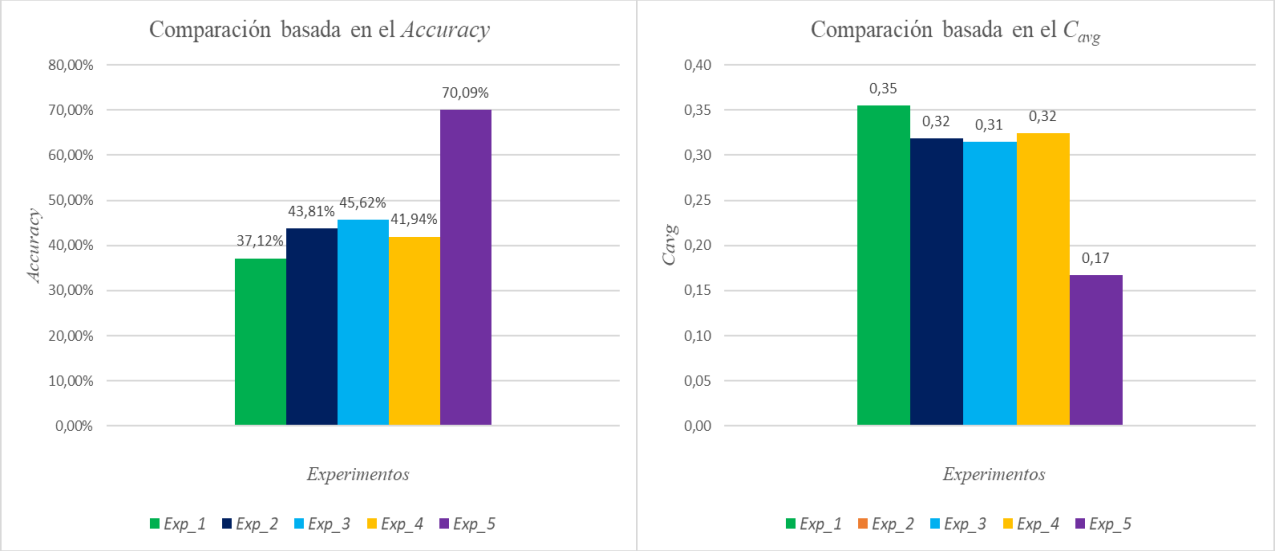


Figura 6-1: Comparación de resultados para todos los experimentos, según los valores del *accuracy* y el C_{avg}.

6.2 Resultados EXP_1

En el *Exp_1* se obtuvo un valor de *accuracy* del 37,12% y del C_{avg} un 0,35 para el mejor modelo. El sistema cuenta con un total aproximado de 810 mil parámetros que se pueden entrenar (tamaño). En la Figura 6-2 se muestran las curvas correspondientes al *accuracy* en los conjuntos de entrenamiento, validación y test.

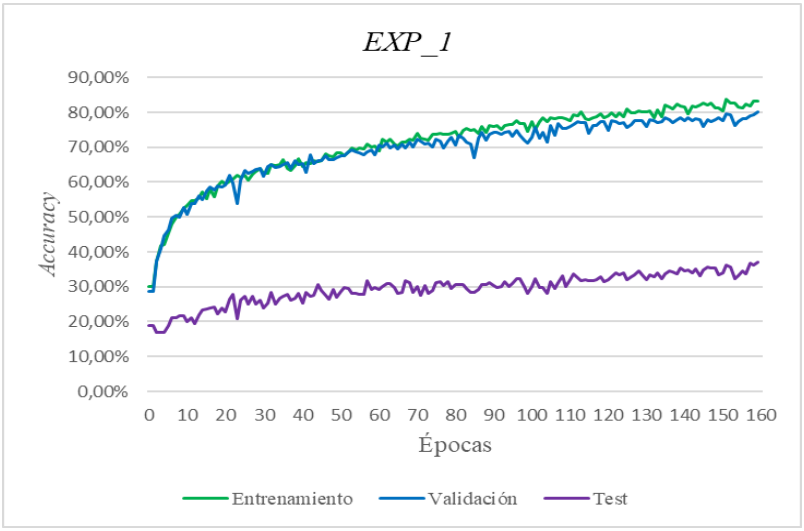


Figura 6-2: Curvas del *accuracy* por cada época.

Cómo se puede observar, el sistema alcanza un rendimiento muy alto en los conjuntos de entrenamiento y de validación. Sin embargo, para el conjunto de test el rendimiento baja considerablemente. Analizando los datos empleados en cada conjunto, se puede encontrar la justificación de este comportamiento. En primer lugar, la red ha sido únicamente entrenada con los datos de entrenamiento y solo ha estado expuesta a los mismos, por lo

tanto, es razonable que obtenga un rendimiento más elevado para este conjunto. En segundo lugar, en el conjunto de validación se obtiene un rendimiento también elevado, pero ligeramente inferior al de entrenamiento a pesar de que estos datos no han sido expuestos en la red durante el entrenamiento. Esto se debe a que el conjunto de validación pertenece al mismo grupo de datos que los de entrenamiento (misma base de datos, pero una partición distinta) y, por lo tanto, los valores de sus correspondientes características son muy similares. Por último, el conjunto de test tampoco ha sido expuesto a la red durante el entrenamiento, sin embargo, a diferencia que los datos del conjunto de entrenamiento y validación, estos son de otro grupo de diferente naturaleza y es por ello por lo que la red no puede identificar el idioma con la misma precisión. El comportamiento anterior ocurre también en el resto de los experimentos, aunque en menor medida en el Exp_5.

6.3 Resultados EXP_2

Siguiendo la metodología expuesta en la sección 4.3, el siguiente paso ha sido aumentar la complejidad de la red y ver el impacto que tiene en el rendimiento. Para ello se utilizaron 512 unidades por cada capa en lugar de las 256 del *Exp_1*.

En el *Exp_2* se obtuvo una mejora del ~15% según el *accuracy* en comparación con el *Exp_1*. En la Figura 6-3 se pueden observar las curvas del *accuracy* por cada época.

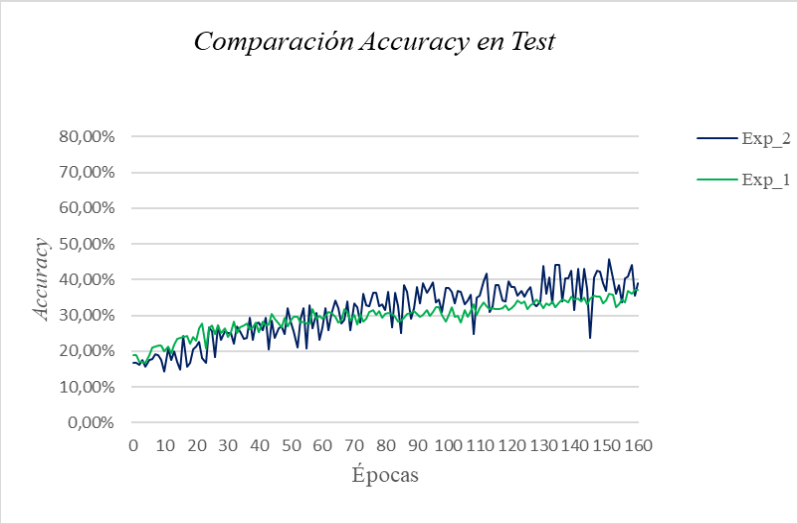


Figura 6-3: Comparación de las curvas del *accuracy* en el conjunto de test por cada época para los *Exp_1* y *Exp_2*.

Un pequeño detalle que se puede observar en el *Exp_2*, es que la curva del *accuracy* tarda cerca de 50 épocas en sobrepasar la curva del *Exp_1*. Un posible motivo por el que puede estar sucediendo este comportamiento es que la red del *Exp_2* cuenta con 512 unidades por cada capa, lo que equivale a aumentar el número de parámetros por 4 veces aproximadamente en comparación con el *Exp_1*. Esto supone que la red tiene que entrenar y adaptar un número de parámetros considerablemente mayor, y, por lo tanto, tarde más al principio en aprender las complejas relaciones entre cada unidad y obtener un mejor rendimiento.

6.4 Resultados EXP_3

El siguiente paso ha sido añadir una capa oculta *feed-forward* al final. En este experimento se obtuvo una mejora del ~4% teniendo en cuenta el valor de *accuracy* del. Nuevamente el sistema logra alcanzar un rendimiento mayor en comparación con el *Exp_2*, aunque esta vez la mejora es muy pequeña. En la Figura 6-4 se pueden ver las correspondientes curvas del valor del *accuracy.*

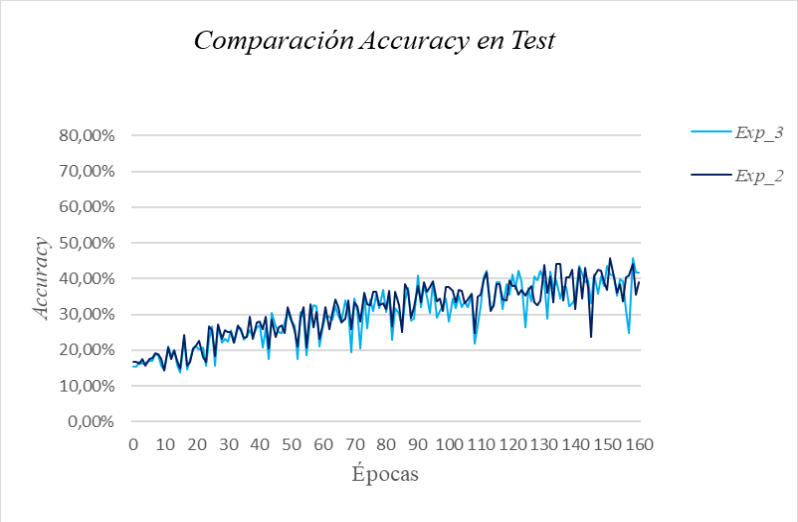


Figura 6-4: Comparación de las curvas del *accuracy* en el conjunto de test por cada época para los *Exp_2* y *Exp_3*.

Cómo se puede apreciar, el comportamiento del *accuracy* es prácticamente igual. Lo que indica que añadir una capa *feed-forward* a continuación de las dos capas ocultas solo ha proporcionado una ligera mejora en el sistema.

6.5 Resultados EXP_4

A diferencia de los experimentos anteriores, en este se han utilizado como características de entrada las derivadas de primer y de segundo orden además de los MFCC. En este experimento se obtuvo un valor de *accuracy* del 41,94%. Esta vez no se ha conseguido mejorar el rendimiento del sistema según el *accuracy* y el $C_{avg.}$

Una posible justificación de la no mejora del rendimiento puede ser el hecho de que, para la red implementada en este experimento en concreto, el añadir información redundante no ayude demasiado a la tarea objetivo, por el contrario, hace ligeramente más difícil la tarea de aprender las complejas relaciones para la arquitectura y características dadas.

La curva del *accuracy* puede ser encontrada en el Anexo, aunque cabe decir que es muy similar a las de los experimentos *Exp_3* y *Exp_2*.

6.6 Resultados EXP_5

Viendo que el mejor rendimiento fue alcanzado a través de la arquitectura implementada en el *Exp_3*, en este experimento se ha decidido utilizar la misma arquitectura, pero esta vez utilizando *bottlenecks* como características de entrada en lugar de MFCC.

En este experimento se obtuvo un valor de *accuracy* del 70,09%, consiguiendo así una mejora del ~36% según el *accuracy* en comparación con el *Exp_3* y logrando aumentar de forma muy considerable el rendimiento del sistema atendiendo tanto al valor del *accuracy* como del C_{avg}.

En la Figura 6-5 se puede observar la curva del *accuracy* en comparación con la del *Exp_3* (mejor resultado con características MFCC).

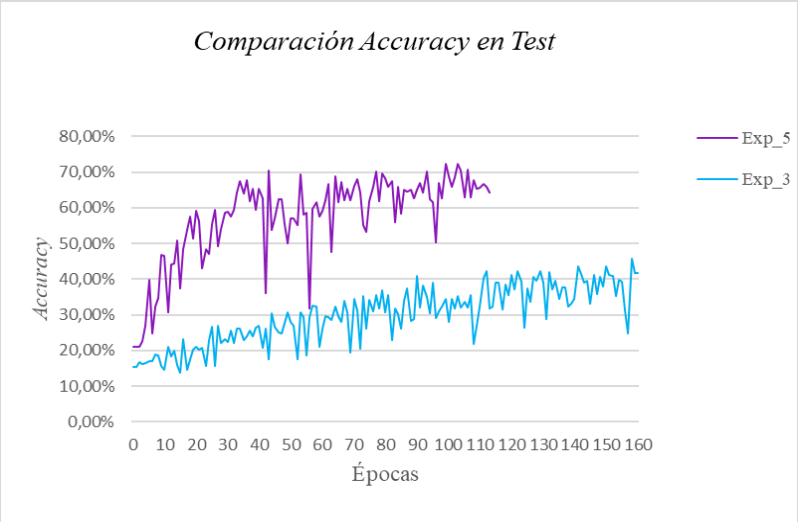


Figura 6-5: Comparación de las curvas del *accuracy* en el conjunto de test por cada época para los *Exp_3* y *Exp_5*.

Como se puede apreciar, el sistema alcanza un rendimiento mucho mayor en un número considerablemente menor de épocas y compuesto por aproximadamente la misma cantidad de parámetros.

La explicación de este resultado como se puede intuir está detrás de las características de entrada. Los *bottlenecks* utilizados fueron extraídos de una red para clasificación fonética, por consiguiente, estas características representan de una mejor forma la información fonética de una lengua. La utilización de dicha información en tareas de reconocimiento de idioma se ha demostrado también en [34] que resulta en un rendimiento considerablemente mayor que el empleo de información acústica de los MFCCs. En [35], resultados experimentales mostraron además que la información fonotáctica es el tipo de información más útil en tareas de discriminación entre idiomas.

Para este experimento final, se verá además la matriz de confusión en la Figura 6-6, la cual nos va a permitir analizar en detalle el rendimiento para cada idioma.

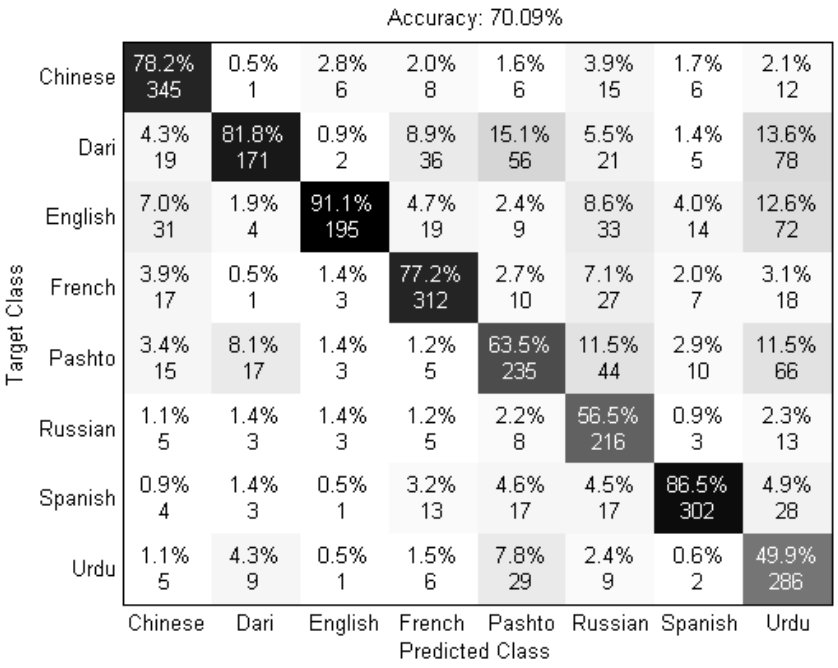


Figura 6-6: Matriz de confusión para el experimento *Exp_5*.

Como se puede ver, la red logra predecir la gran mayoría de las secuencias para los idiomas *English* y *Spanish* mientras que para los idiomas *Russian* y *Urdu* presenta aún múltiples fallos en las predicciones.

7 Conclusiones y trabajo futuro

En este trabajo, se ha propuesto una comparación del rendimiento entre diferentes modelos basados redes neuronales compuestas principalmente por arquitecturas LSTMs en la tarea del reconocimiento de idioma a través de secuencias de 3 segundos extraídas de la base de datos de la evaluación NIST LRE 2009. Motivado por la utilización de las redes neuronales en este campo, se han explorado diferentes arquitecturas que toman como características de entrada MFCC, MFCC + Δ + ΔΔ y *bottlenecks* extraídos de una red entrenada para clasificación de unidades fonéticas.

El aspecto más relevante que muestran los diferentes experimentos es que las redes compuestas principalmente por capas LSTMs presentan un rendimiento considerablemente mayor cuando utilizan como características de entrada *bottlenecks* que contienen información fonética de la señal de voz en comparación a cuando utilizan MFCC o MFCC + Δ + ΔΔ, teniendo en cuenta las medidas de *accuracy* y *Cavg*. En concreto, se logró con la misma arquitectura una mejora del ~36% de *accuracy* utilizando características *bottleneck.*

Otros puntos relevantes extraídos de los experimentos han sido:

- Duplicar las unidades de las capas ocultas propicia que las redes neuronales sean capaces de aprender relaciones más complejas y por lo tanto obtener un mejor rendimiento.
- La utilización de características MFCC + Δ + ΔΔ no ha presentado ninguna mejora en comparación con los MFCC para el problema tratado en este trabajo y las arquitecturas empleadas.
- Aunque todos los segmentos utilizados han sido de voz, la naturaleza de los datos con los que se entrena la red tiene un gran impacto en el rendimiento de esta cuando se utilizan MFCC como características de entrada.
- La combinación de capas LSTMs con una *fully-connected* al final presenta una cierta mejora en la tarea objetivo de los experimentos.
- El mejor sistema propuesto ha logrado una mejora del ~15% en comparación con uno de los sistemas del estado del arte basado en *iVectors*, teniendo en cuenta el C_{avg}.

Para intentar mejorar el rendimiento conseguido en el mejor de los experimentos llevados a cabo, se proponen como trabajo futuro los siguientes pasos:

- Añadir más capas ocultas en las arquitecturas y ver el impacto que tiene en el rendimiento del modelo.
- Establecer diferentes configuraciones de los hiperparámetros.
- Evaluar el rendimiento del último sistema en secuencias de diferente duración y adaptar la red a los resultados.
- Indagar en detalle el motivo por el cual puede el rendimiento de la red sea diferente para cada idioma.
- Aplicar otro tipo de redes neuronales, como pueden ser las CNNs o las RNNs.
- Añadir más idiomas objetivo.

Referencias

[1] Torres-Carrasquillo, P.A., Singer, E., Kohler, M.A., Greene, R.J., Reynolds, D.A., & Deller, J.R (2002). "Approaches to Language Identification using Gaussian Mixture Models and Shifted Delta Cepstral Features" *7th International Conference on Spoken Language Processing*.

[2] Zissman, M. A. and Berkling, K. M. (2001). "Automatic language identification". *Speech Communication*, vol. 35, pp. 115-124.

[3] Bartz, C., Herold, T., Yang, H., & Meinel, C. (2017). "Language Identification Using Deep Convolutional Recurrent Neural Networks". *International Conference on Neural Information Processing*, pp. 880-889.

[4] Zazo, R., Lozano-Diez, A., González-Domínguez, J., Toledano, D. y González-Rodríguez, J. (2016). "Language identification in short utterances using long short-term memory (lstm) recurrent neural networks," *PloS one*, vol.11, no.1.

[5] Tiwari, V. (2010). "MFCC and its applications in speaker recognition". *International Journal on Emerging Technologies*, vol.1, no.1, pp. 19-22.

[6] Dave, N. (2013). "Feature Extraction Methods LPC, PLP and MFCC in Speech Recognition*", International journal for advance research in engineering and technology*, vol.1, no.6, pp. 1-4.

[7] Benesty, J., Sondhi, M. M., & Huang, Y. (2007). "*Springer handbook of speech processing",* Chapter 4: Shifted Delta Cepstral Features. Springer.

[8] Reynolds, D.A., Quatieri, T. F. and Dunn, R. (2000). "Speaker verification using adapted Gaussian mixture models". *Digital Signal Processing*, vol.10, no.1-3, pp. 19-41.

[9] Reynolds, D. A. (1995) "Speaker identification and verification using Gaussian mixture speaker models". *Speech Commun*. vol.17, pp. 91–108.

[10] González-Rodríguez, J. "Reconocimiento Automático de Locutor", *ATVS–Biometric Recognition Group*. Escuela Politéctina Superior, Universidad Autónoma de Madrid.

[11] You, C. H., Li, H. and Lee, K. A. (2010) ''A GMM-supervector approach to language recognition with adaptive relevance factor,'' *Signal Processing Conference, 2010 18th European*, p. 1993-1997.

[12] González-Domínguez, J., López-Moreno, I., Sak, H., González-Rodríguez, J. and Moreno, P. (2014). "Automatic language identification using long short-term memory recurrent neural networks". *Fifteenth Annual Conference of the International Speech Communication Association.*

[13] García-Romero, D. and Espy-Wilson, C. Y. (2011). “Analysis of ivector length normalization in speaker recognition systems” *Twelfth Annual Conference of the International Speech Communication Association.*

[14] Dehak, N., Kenny, P., Dehak, R., Dumouchel, P. and Ouellet, P. (2010). "Front-End Factor Analysis for Speaker Verification". *IEEE Transactions on Audio, Speech and Language Processing*, vol. 19, no. 4, pp. 788-798.

[15] Dehak, N., Torres-Carrasquillo, P. A., Reynolds, D. A. and Dehak, R. (2011) “Language Recognition via i-vectors and Dimensionality Reduction”. *Twelfth annual conference of the international speech communication association.*

[16] Haykin, S. “*Neural Networks and Learning Machines*”, Third Edition, Chapter 1.1, pp. 2.

[17] https://deeplearning4j.org/neuralnet-overview#introduction-to-deep-neuralnetworks-deep-learning

[18] Karlik, B., and Olgac, A.V. (2010). “Performance analysis of various activation functions in generalized MLP architectures of neural networks *International Journal of Artificial Intelligence and Expert Systems,* 2011, vol. 1, no 4, p. 111-122.

[19] Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986). “Learning representations by back-propagating errors”. *Nature,* vol.323, no 6088, pp. 533-536.

[20] http://colah.github.io/posts/2015-08-Backprop/

[21] Tim Jones, M. (2017). *Deep learning architectures*. Descargado el 27 de marzo de 2018 de https://www.ibm.com/developerworks/library/cc-machine-learning-deep-learning-architectures/index.html

[22] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). “On the difficulty of training recurrent neural networks”. *International Conference on Machine Learning,* pp.1310-1318.

[23] Hochreiter, S., and Schmidhuber, J. (1997). “Long short-term memory”. *Neural computation,* vol. 9, no.8, pp. 1735-1780.

[24] Sundermeyer, M. Schluter, R. and Ney, H. (2012). “Lstm neural networks for language modeling”. *Thirteenth Annual Conference of the International Speech Communication Association*

[25] Graves, A., and Schmidhuber, J. (2005). “Framewise phoneme classification with bidirectional LSTM and other neural network architectures”. *Neural Networks,* vol.1, no.5-6, pp.602-610.

[26] Matejka, P., Zhang, L., Ng, T., Mallidi, H. S., Glembek, O., Ma, J., and Zhang, B. (2014). “Neural network bottleneck features for language identification”. *Proc. IEEE Odyssey*, pp. 299-304.

[27] López-Moreno, I, González-Domínguez, J., Plchot, O., Martínez, D., González-Rodríguez, J. and Moreno, P. (2014). "Automatic language identification using deep neural networks," *Acoustics, Speech and Signal Processing (ICASSP), IEEE International Conference*, pp. 5374–5378.

[28] Lozano-Díez, A., Zazo-Candil, R., González-Domínguez, J., Toledano, D. T., and González-Rodríguez, J. (2015). "An end-to-end approach to language identification in short utterances using convolutional neural networks". *Sixteenth Annual Conference of the International Speech Communication Association.*

[29] Gelly, G., Gauvain, J. L., Le, V. B., and Messaoudi, A. (2016). "A Divide-and-Conquer Approach for Language Identification Based on Recurrent Neural Networks". *Interspeech,* pp. 3231-3235.

[30] https://keras.io

[31] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C. ... & Ghemawat, S. (2016). "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". *ArXiv preprint arXiv:1603.04467*.

[32] The 2007 NIST Language Recognition Evaluation Plan (LRE07).

[33] http://neuralnetworksanddeeplearning.com/chap3.html#the_cross-entropy_cost_function

[34] Tang, Z., Wang, D., Chen, Y., Li, L., and Abel, A. (2018). "Phonetic temporal neural model for language identification". *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol.26, no.1, pp. 134-144.

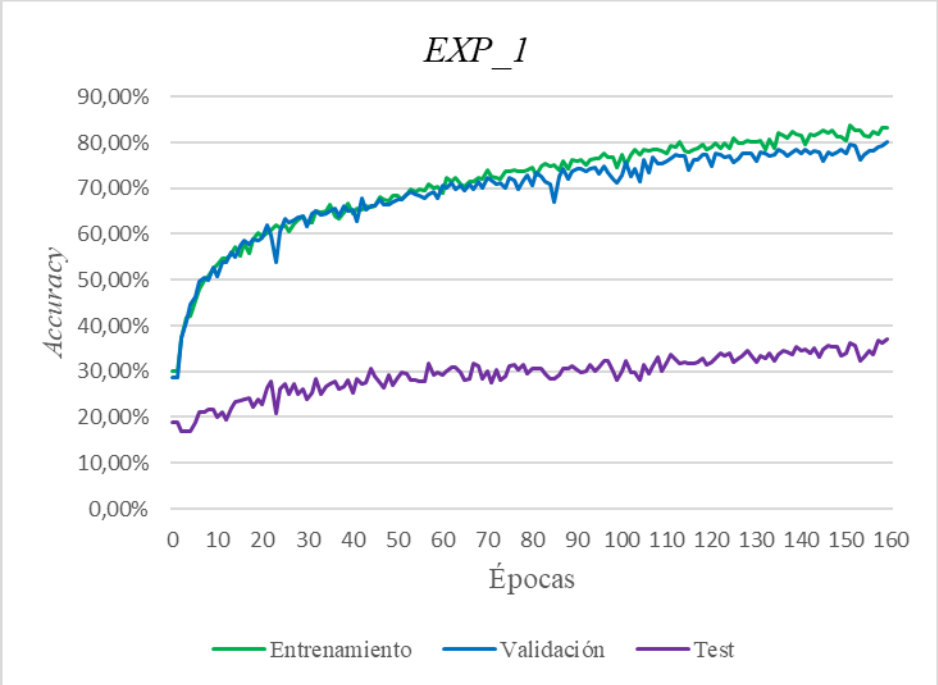
[35] Hazen, T. J., and Zue, V. W. (1997). "Segment-based automatic language identification". *The Journal of the Acoustical Society of America*, vol.101, no.4, pp. 2323-2331.

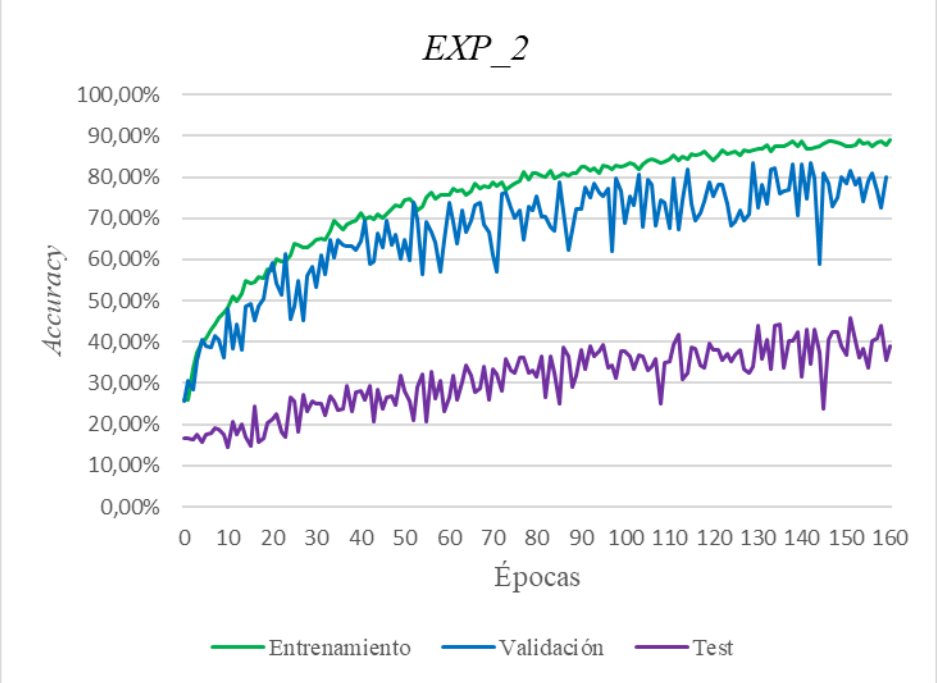
Glosario

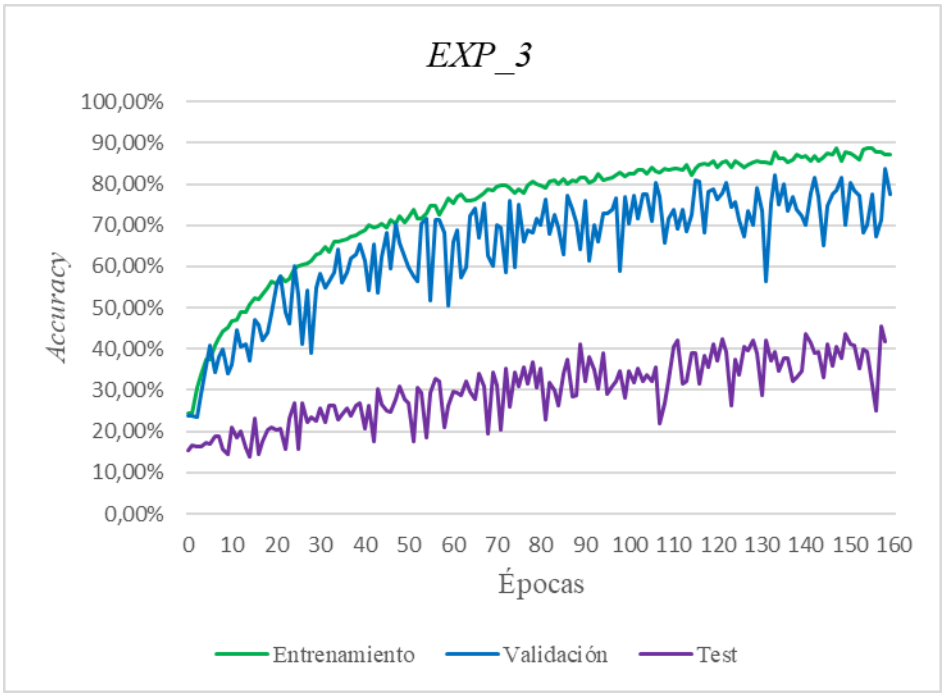
LSTM: *Long Short Term Memory*
LID: *Language Identification*
MFCC: *Mel Frequency Cepstral Coefficients*
FFT: *Fast Fourier Transform*
DCT*: Discrete Cosine Transform*
GMMs: *Gaussian Mixture Models*
UBM: *Universal Background Model*
MAP: *Máximo a Posteriori*
JFA: *Join Factor Analysis*
MSE: *Mean Squared Error*
RNN: *Recurrent Neural Networks*
CNN: *Convolutional Neural Networks*
DBN: *Deep belief networks*
DSN*: Deep stacking networks*
GRU: *Gated Recurrent Unit*
NPL: *Natural Language Processing*
BN-NN: 3 *Bottleneck Neural-Network*
SDC: *Shifted Delta Coefficients*
DNN: *Deep Neural Network*
PLP: *Perceptual Linear Prediction*
GPU: *Graphics Processing Unit*
NIST: *National Institute of Standards and Technology*
LRE: *Language Recognition Evaluation*
VOA: *Voice of America*
CTS: *Conversational Telephone Speech*
Cavg: *Media del coste*
ASR: *Automatic Speech Recognition*
CPU*: Central Processing Unit*

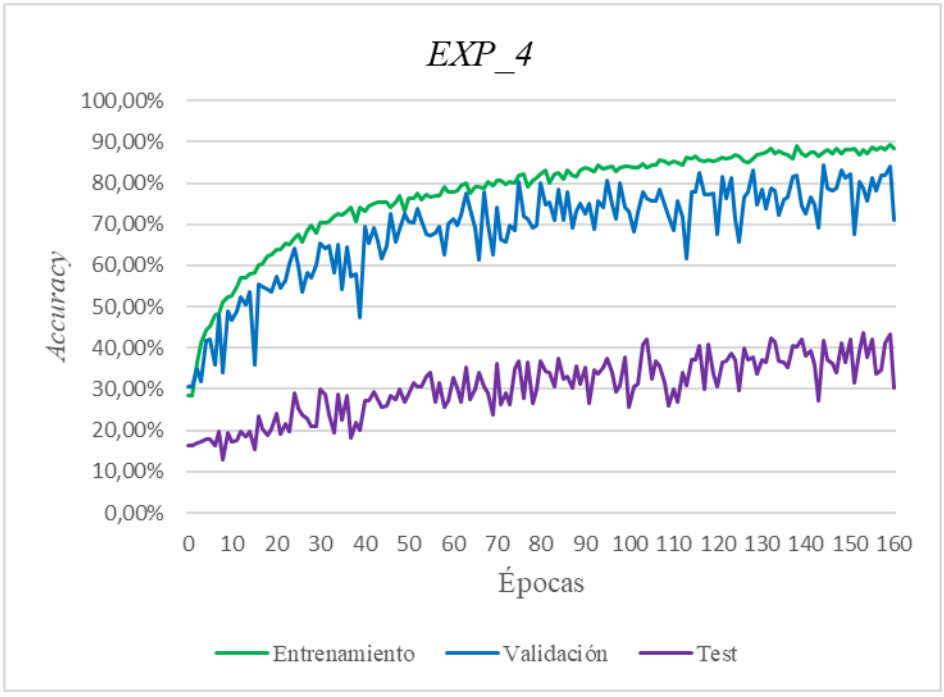
Anexos

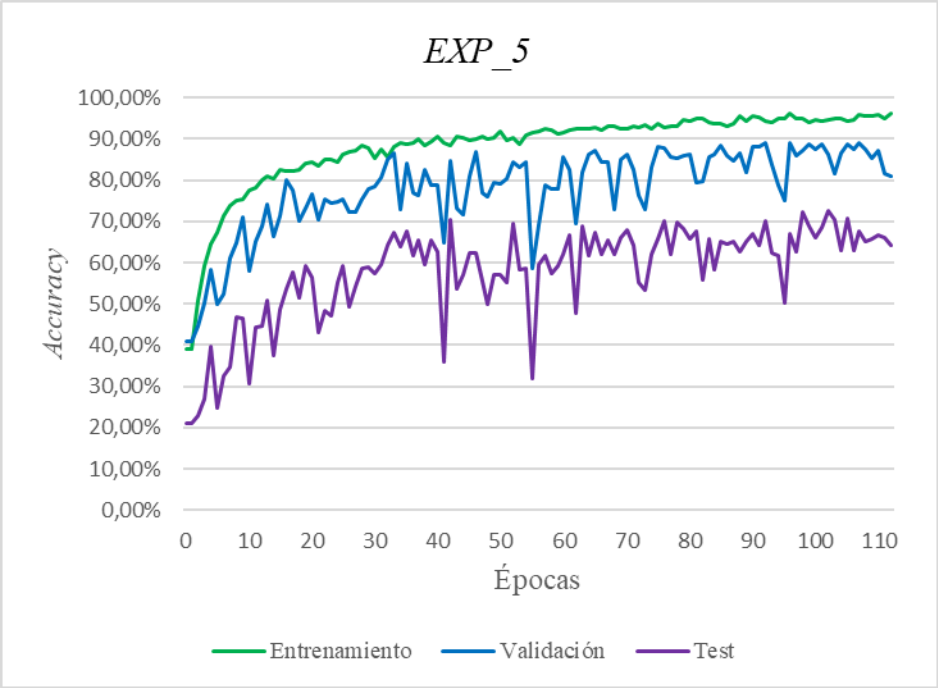
A Curvas del accuracy para cada experimento






EXP_3
100,00%
90,00%
80,00%
70,00%
60,00%
50,00%
40,00%
30,00%
20,00%
10,00%
0,00%
Accuracy
0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160
Épocas
Entrenamiento
Validación
Test

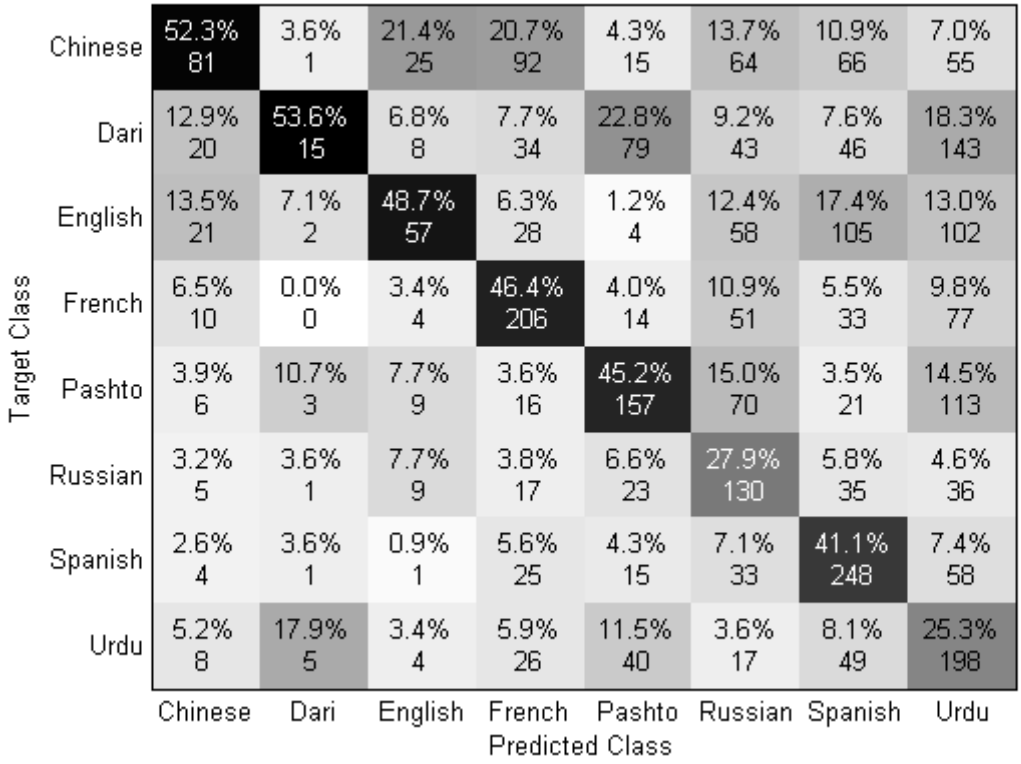

EXP_4
100,00%
90,00%
80,00%
70,00%
60,00%
50,00%
40,00%
30,00%
20,00%
10,00%
0,00%
Accuracy
0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160
Épocas
Entrenamiento
Validación
Test


EXP_5
100,00%
90,00%
80,00%
70,00%
60,00%
50,00%
40,00%
30,00%
20,00%
10,00%
0,00%
Accuracy
0
10
20
30
40
50
60
70
80
90
100
110
Épocas
Entrenamiento
Validación
Test

B Matrices de confusión para cada experimento

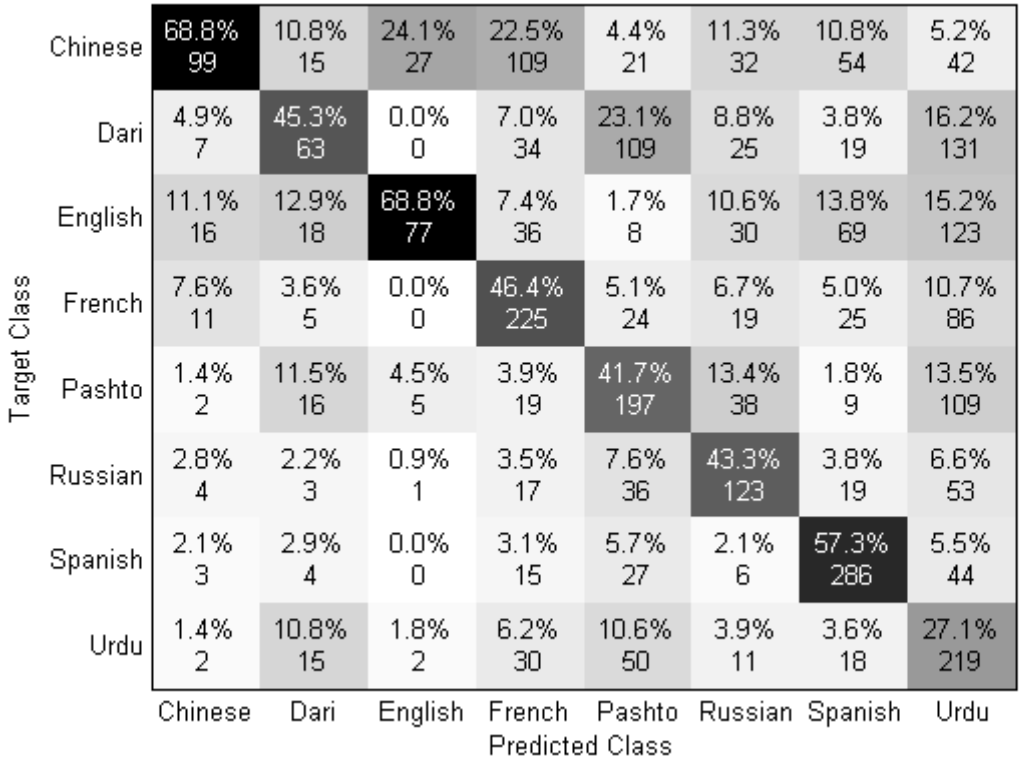
Exp_1

Accuracy: 37.12%

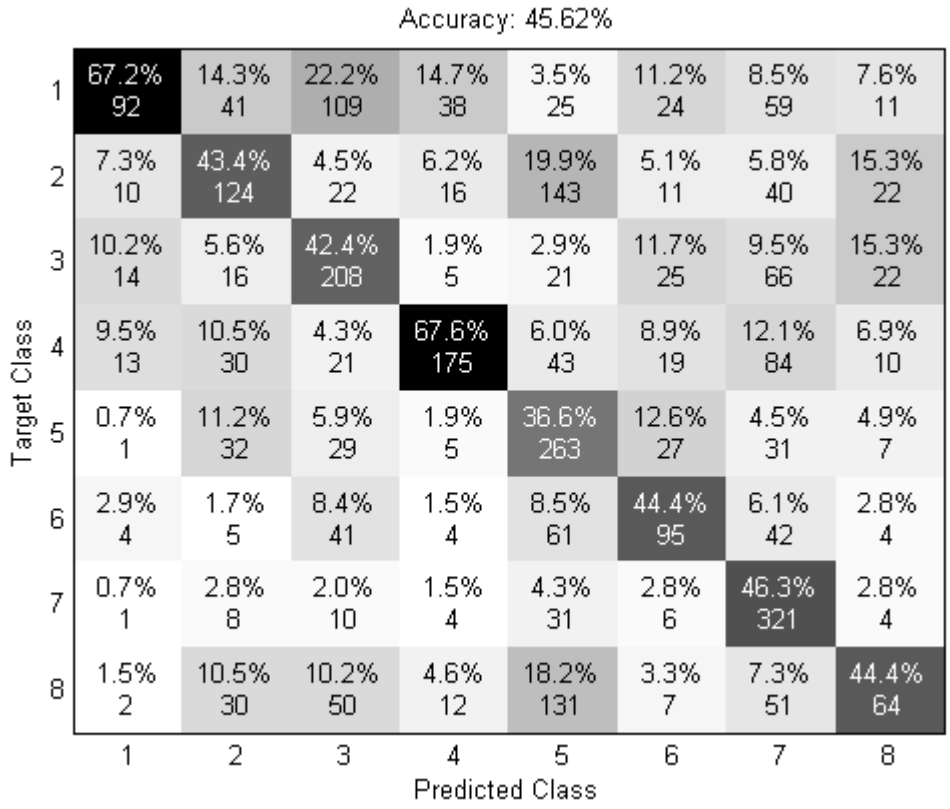


Exp_2

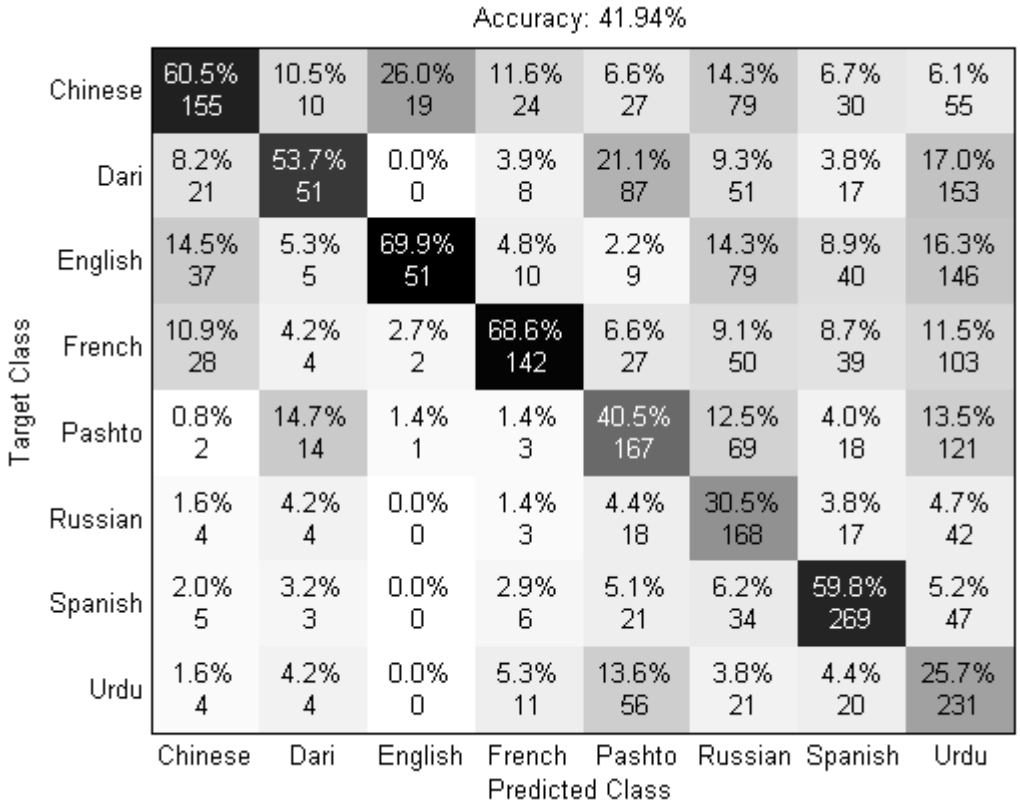
Accuracy: 43.81%



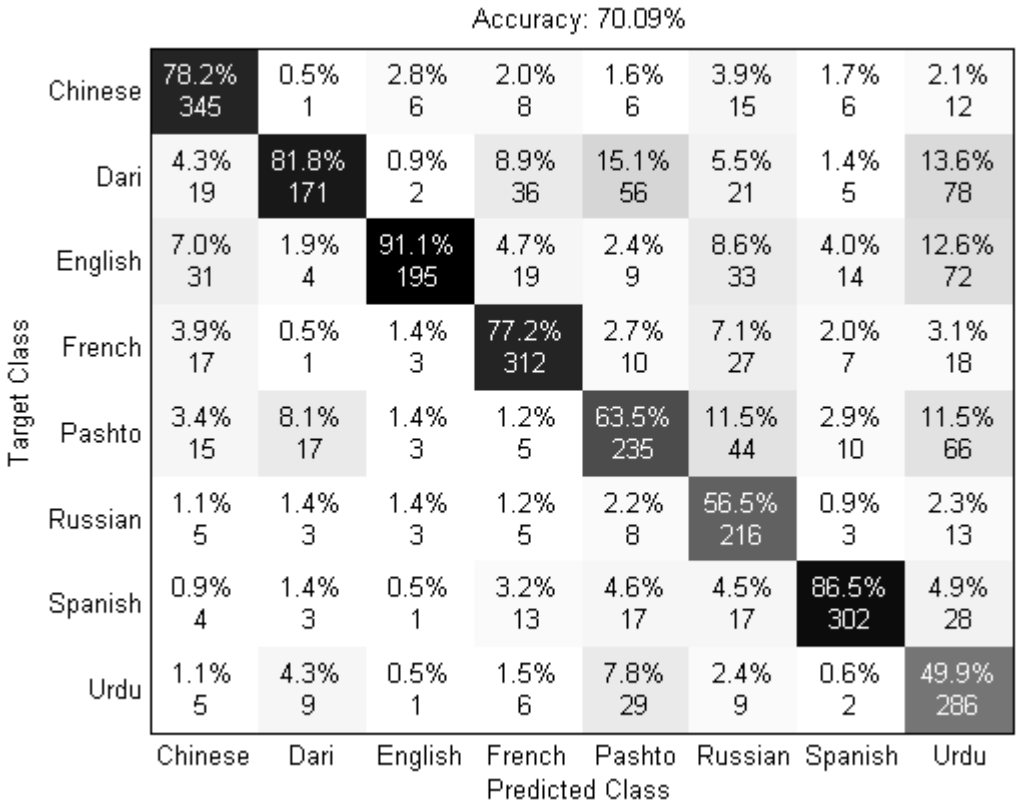
Exp_3


Accuracy: 45.62%
Target Class
Predicted Class
1 67.2% 92 14.3% 41 22.2% 109 14.7% 38 3.5% 25 11.2% 24 8.5% 59 7.6% 11
2 7.3% 10 43.4% 124 4.5% 22 6.2% 16 19.9% 143 5.1% 11 5.8% 40 15.3% 22
3 10.2% 14 5.6% 16 42.4% 208 1.9% 5 2.9% 21 11.7% 25 9.5% 66 15.3% 22
4 9.5% 13 10.5% 30 4.3% 21 67.6% 175 6.0% 43 8.9% 19 12.1% 84 6.9% 10
5 0.7% 1 11.2% 32 5.9% 29 1.9% 5 36.6% 263 12.6% 27 4.5% 31 4.9% 7
6 2.9% 4 1.7% 5 8.4% 41 1.5% 4 8.5% 61 44.4% 95 6.1% 42 2.8% 4
7 0.7% 1 2.8% 8 2.0% 10 1.5% 4 4.3% 31 2.8% 6 46.3% 321 2.8% 4
8 1.5% 2 10.5% 30 10.2% 50 4.6% 12 18.2% 131 3.3% 7 7.3% 51 44.4% 64
1 2 3 4 5 6 7 8

Exp_4


Accuracy: 41.94%
Target Class
Predicted Class
Chinese 60.5% 155 10.5% 10 26.0% 19 11.6% 24 6.6% 27 14.3% 79 6.7% 30 6.1% 55
Dari 8.2% 21 53.7% 51 0.0% 0 3.9% 8 21.1% 87 9.3% 51 3.8% 17 17.0% 153
English 14.5% 37 5.3% 5 69.9% 51 4.8% 10 2.2% 9 14.3% 79 8.9% 40 16.3% 146
French 10.9% 28 4.2% 4 2.7% 2 68.6% 142 6.6% 27 9.1% 50 8.7% 39 11.5% 103
Pashto 0.8% 2 14.7% 14 1.4% 1 1.4% 3 40.5% 167 12.5% 69 4.0% 18 13.5% 121
Russian 1.6% 4 4.2% 4 0.0% 0 1.4% 3 4.4% 18 30.5% 168 3.8% 17 4.7% 42
Spanish 2.0% 5 3.2% 3 0.0% 0 2.9% 6 5.1% 21 6.2% 34 59.8% 269 5.2% 47
Urdu 1.6% 4 4.2% 4 0.0% 0 5.3% 11 13.6% 56 3.8% 21 4.4% 20 25.7% 231
Chinese Dari English French Pashto Russian Spanish Urdu

Exp_5


Accuracy: 70.09%
Target Class
Chinese
Dari
English
French
Pashto
Russian
Spanish
Urdu
78.2% 345
0.5% 1
2.8% 6
2.0% 8
1.6% 6
3.9% 15
1.7% 6
2.1% 12
4.3% 19
81.8% 171
0.9% 2
8.9% 36
15.1% 56
5.5% 21
1.4% 5
13.6% 78
7.0% 31
1.9% 4
91.1% 195
4.7% 19
2.4% 9
8.6% 33
4.0% 14
12.6% 72
3.9% 17
0.5% 1
1.4% 3
77.2% 312
2.7% 10
7.1% 27
2.0% 7
3.1% 18
3.4% 15
8.1% 17
1.4% 3
1.2% 5
63.5% 235
11.5% 44
2.9% 10
11.5% 66
1.1% 5
1.4% 3
1.4% 3
1.2% 5
2.2% 8
56.5% 216
0.9% 3
2.3% 13
0.9% 4
1.4% 3
0.5% 1
3.2% 13
4.6% 17
4.5% 17
86.5% 302
4.9% 28
1.1% 5
4.3% 9
0.5% 1
1.5% 6
7.8% 29
2.4% 9
0.6% 2
49.9% 286
Chinese
Dari
English
French
Pashto
Russian
Spanish
Urdu
Predicted Class